INFO510 : Algorithmique et structures de données complexes

SCIENCES
ET MONTAGNE
UNIVERSITÉ SAVOIE
MONT BLANC

2017-2018

Organisation du module

- Répartition des enseignements :
  - 7 CM (1h30), 6 TD (1h30), 3 TP (4h).

- Intervenants :
  - CM : Clovis Eberhart
    `clovis.eberhart@univ-smb.fr`
  - TD : Clovis Eberhart, Gérald Cavallini
    `gerald.cavallini@univ-smb.fr`
  - TP : Clovis Eberhart, Gérald Cavallini

Ressources

- Ressources informatiques :
  - www.lama.univ-savoie.fr/wiki
    - Polycopié INFO510.
    - Énoncés des TD et TP.
    - Polycopiés de cours similaires (rédigés par d'autres enseignants).

Ressources

- Ressources informatiques :
  - www.lama.univ-savoie.fr/wiki
    - Polycopié INFO510.
    - Énoncés des TD et TP.
    - Polycopiés de cours similaires (rédigés par d'autres enseignants).
- Ressources bibliographiques :
  - Cormen, Leiserson et Rivest, *Introduction à l'algorithmique* (2004).
  - Kernighan et Ritchie, *The C programming language* (1978).
  - Braquelaire, *Méthodologie de la programmation en C* (2005).
  - Malgouyres, Zrour et Feschet, *Initiation à l'algorithmique et à la programmation en C* (2010).

Ressources

- Ressources informatiques :
  - www.lama.univ-savoie.fr/wiki
    - Polycopié INFO510.
    - Énoncés des TD et TP.
    - Polycopiés de cours similaires (rédigés par d'autres enseignants).
- Ressources bibliographiques :
  - Cormen, Leiserson et Rivest, *Introduction à l'algorithmique* (2004).
  - Kernighan et Ritchie, *The C programming language* (1978).
  - Braquelaire, *Méthodologie de la programmation en C* (2005).
  - Malgouyres, Zrour et Feschet, *Initiation à l'algorithmique et à la programmation en C* (2010).
- Ressources humaines
  - Les intervenants : Gérald et moi !

Definition (Algorithme)

Ensemble d'opérations dont l'enchaînement permet de résoudre un problème en un nombre fini d'étapes.

Definition (Algorithme)

Ensemble d'opérations dont l'enchaînement permet de résoudre un problème en un nombre fini d'étapes.

- Du nom mathématicien/astronome perse **Muhammad ibn Musa al-Khwarizmi** (Ouzbekistan 780 – Bagdad 850). (Également auteur du livre *Abrégé du calcul par la restauration et la comparaison*,
Kitab al-mukhtasar fi hisab al-jabr wa-l-muqabala)

Definition (Algorithme)

Ensemble d'opérations dont l'enchaînement permet de résoudre un problème en un nombre fini d'étapes.

- Du nom mathématicien/astronome perse **Muhammad ibn Musa al-Khwarizmi** (Ouzbekistan 780 – Bagdad 850). (Également auteur du livre *Abrégé du calcul par la restauration et la comparaison*,
Kitab al-mukhtasar fi hisab **al-jabr** *wa-l-muqabala*)

Definition (Algorithme)

Ensemble d'opérations dont l'enchaînement permet de résoudre un problème en un nombre fini d'étapes.

- Du nom mathématicien/astronome perse **Muhammad ibn Musa al-Khwarizmi** (Ouzbekistan 780 – Bagdad 850). (Également auteur du livre *Abrégé du calcul par la restauration et la comparaison*,
Kitab al-mukhtasar fi hisab **al-jabr** *wa-l-muqabala*)

- Chacune des opérations d'un algorithme est composée d'une ou plusieurs opérations élémentaires.

Definition (Algorithme)

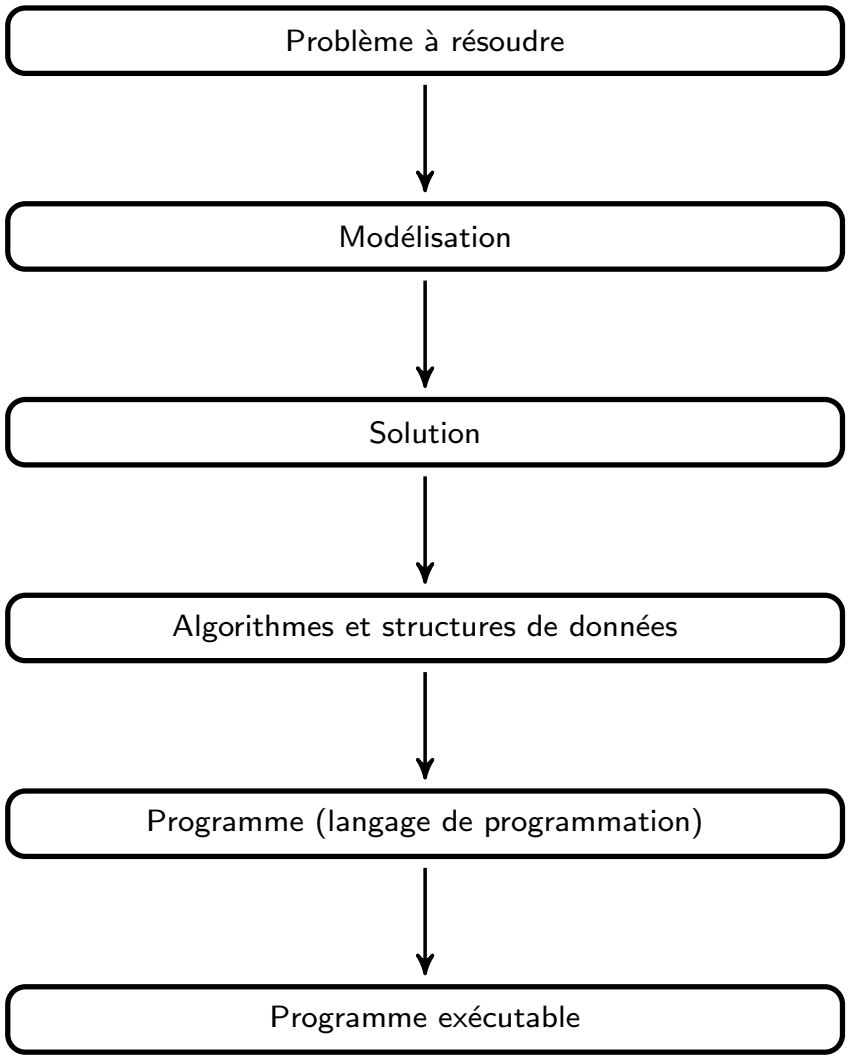
Ensemble d'opérations dont l'enchaînement permet de résoudre un problème en un nombre fini d'étapes.

- Du nom mathématicien/astronome perse **Muhammad ibn Musa al-Khwarizmi** (Ouzbekistan 780 – Bagdad 850). (Également auteur du livre *Abrégé du calcul par la restauration et la comparaison*,
Kitab al-mukhtasar fi hisab **al-jabr** *wa-l-muqabala*)

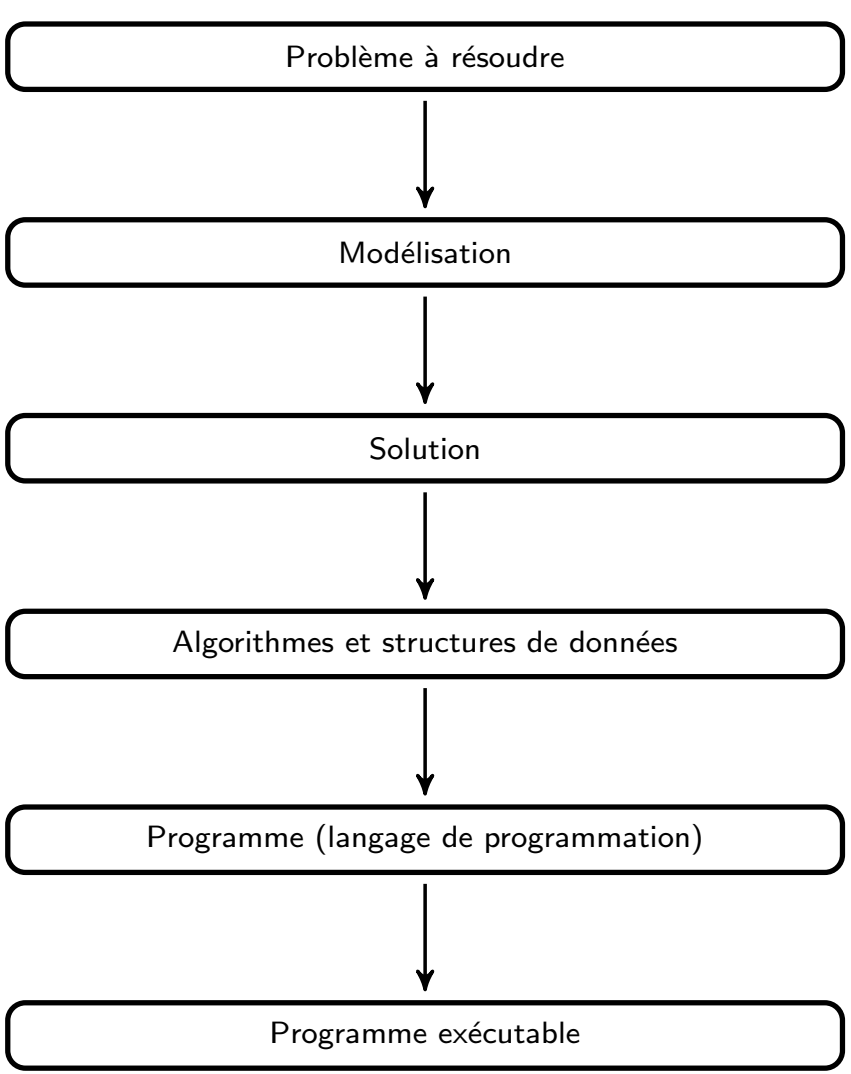
- Chacune des opérations d'un algorithme est composée d'une ou plusieurs opérations élémentaires.

- Un algorithme doit toujours se terminer en un temps fini.

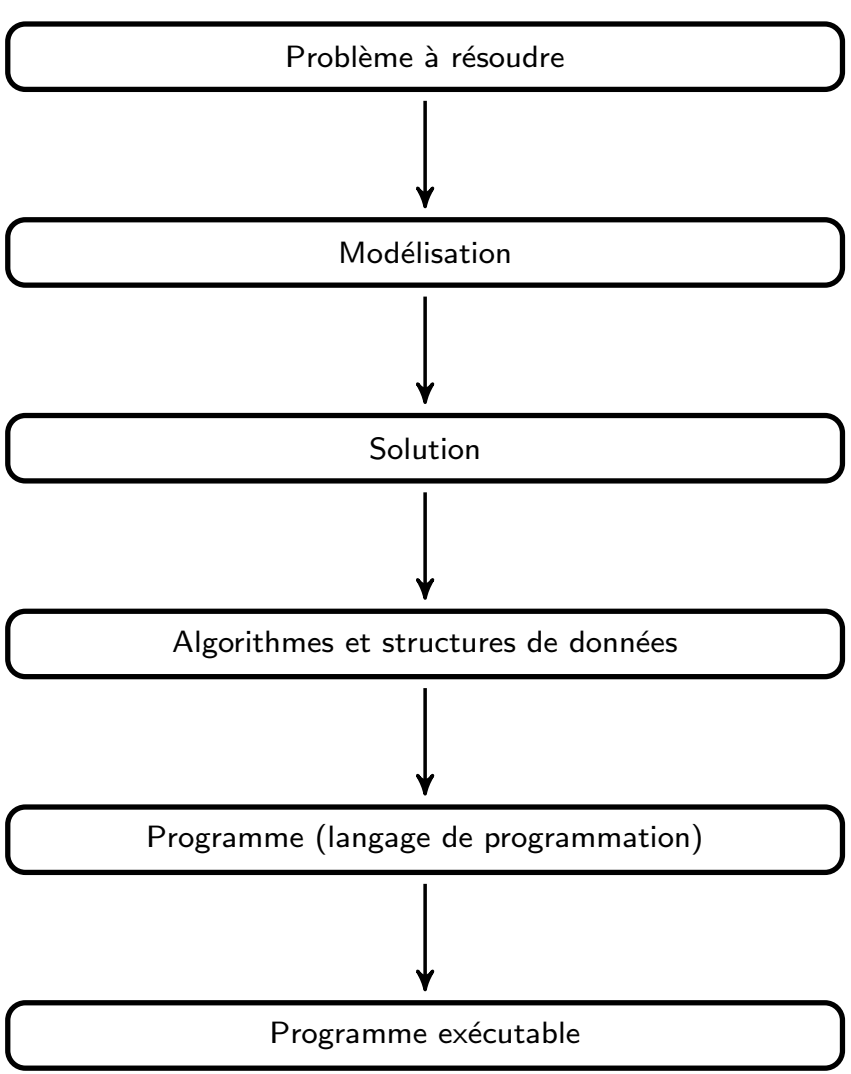
Analyse, algorithmique et programmation



Analyse, algorithmique et programmation



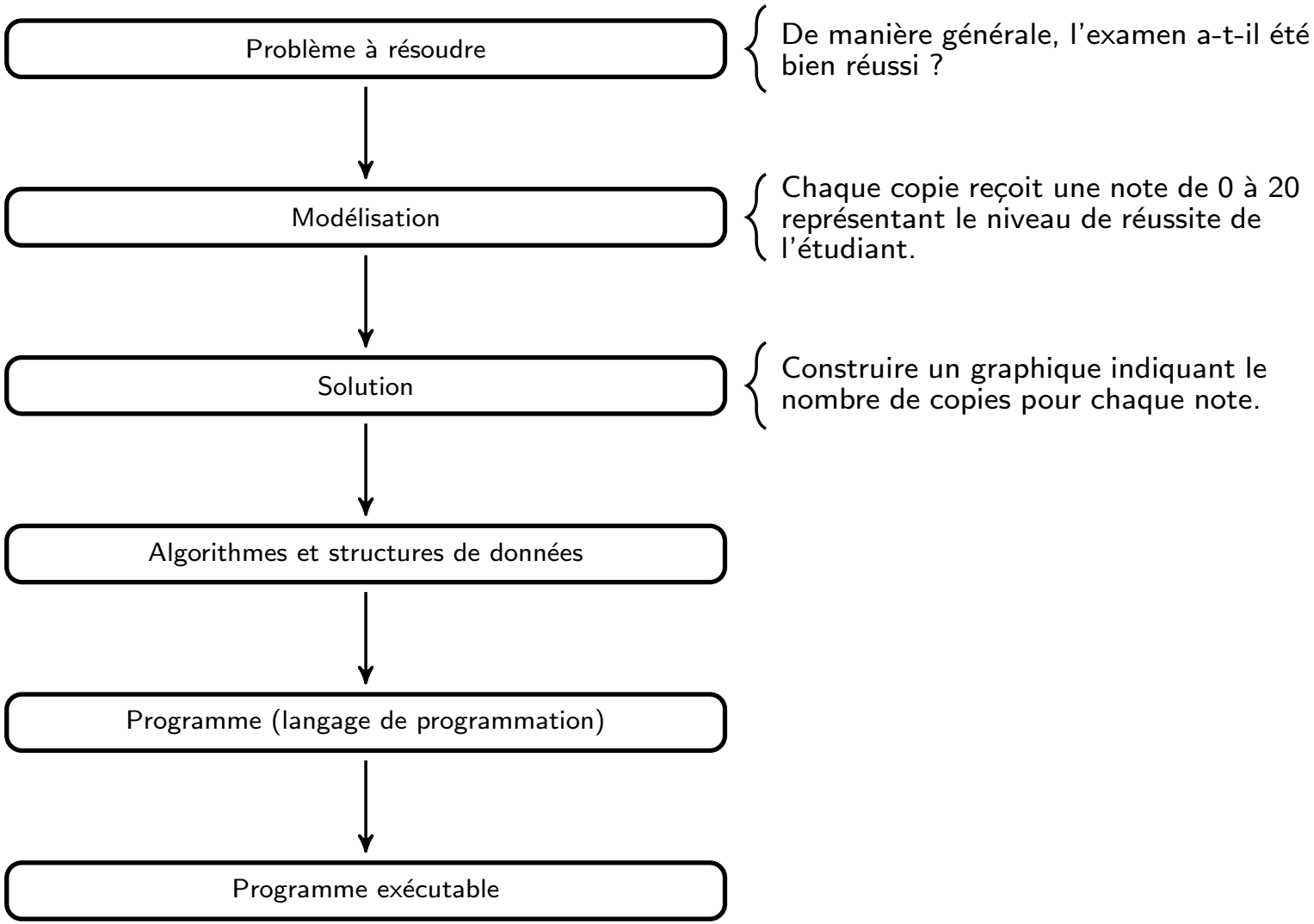
De manière générale, l'examen a-t-il été bien réussi ?



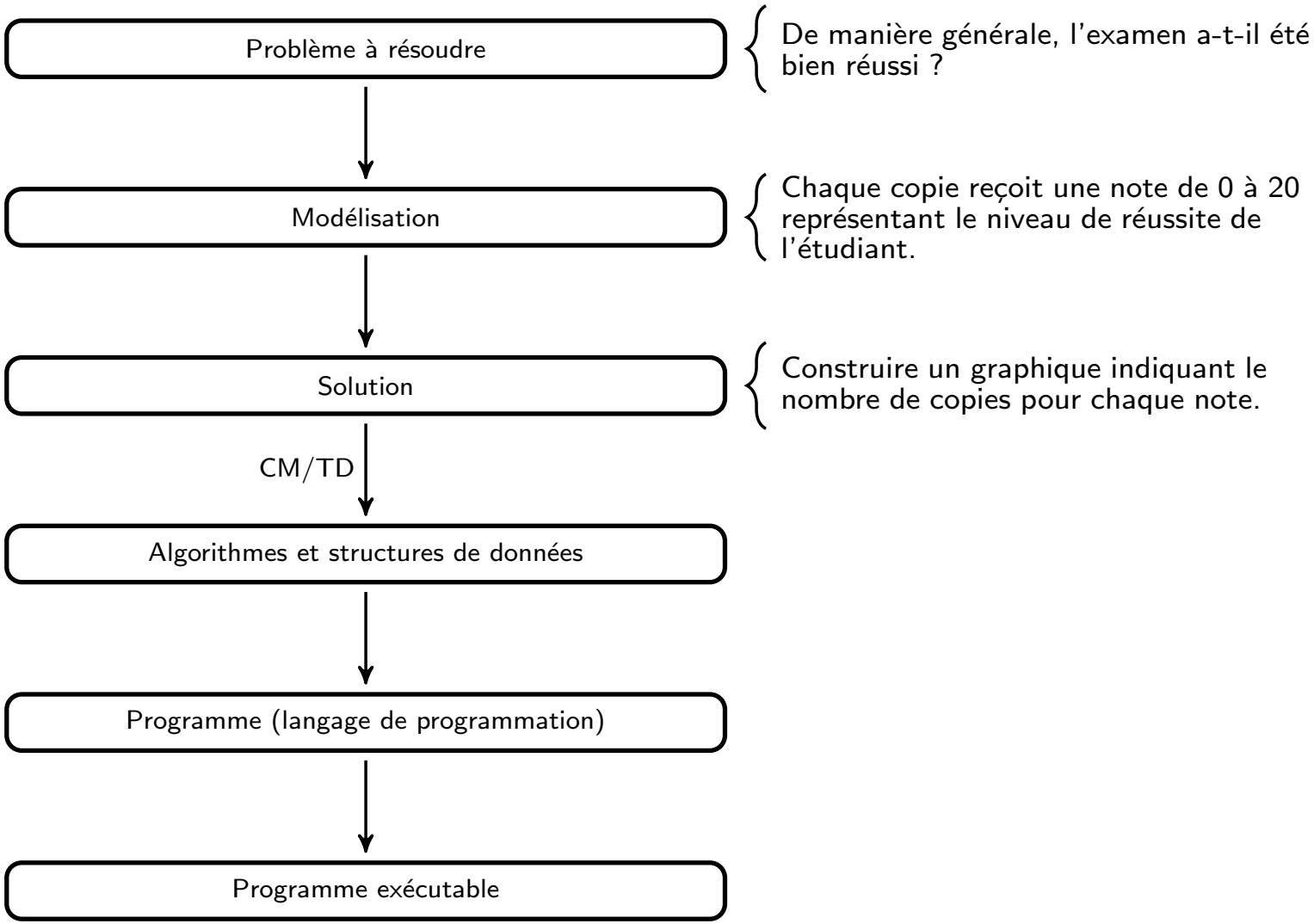
De manière générale, l'examen a-t-il été bien réussi ?

Chaque copie reçoit une note de 0 à 20 représentant le niveau de réussite de l'étudiant.

Analyse, algorithmique et programmation



Analyse, algorithmique et programmation



Pour chaque note de 0 à 20, compter le nombre de copies ayant cette note
Afficher le résultat sous la forme d'un graphique à barres

Pour chaque note de 0 à 20, compter le nombre de copies ayant cette note
Afficher le résultat sous la forme d'un graphique à barres

Définir un compteur pour chaque valeur de 0 à 20.
$\longrightarrow$ Pour chaque copie, incrémenter le compteur correspondant à sa note.
Afficher le résultat sous la forme d'un graphique à barres

Pour chaque note de 0 à 20, compter le nombre de copies ayant cette note
Afficher le résultat sous la forme d'un graphique à barres

Définir un compteur pour chaque valeur de 0 à 20.
$\longrightarrow$ Pour chaque copie, incrémenter le compteur correspondant à sa note.
Afficher le résultat sous la forme d'un graphique à barres

Definir un tableau à 21 cases.
Initialiser toutes ces cases à 0.
$\longrightarrow$
Pour chaque copie, incrémenter la case correspondante à sa note.
Afficher le résultat sous la forme d'un graphique à barres

Pour chaque note de 0 à 20, compter le nombre de copies ayant cette note
Afficher le résultat sous la forme d'un graphique à barres

Définir un compteur pour chaque valeur de 0 à 20.
$\longrightarrow$ Pour chaque copie, incrémenter le compteur correspondant à sa note.
Afficher le résultat sous la forme d'un graphique à barres

Definir un tableau à 21 cases.
Initialiser toutes ces cases à 0.
$\longrightarrow$
Pour chaque copie, incrémenter la case correspondante à sa note.
Afficher le résultat sous la forme d'un graphique à barres

$\vdots$

Pour chaque note de 0 à 20, compter le nombre de copies ayant cette note
Afficher le résultat sous la forme d'un graphique à barres

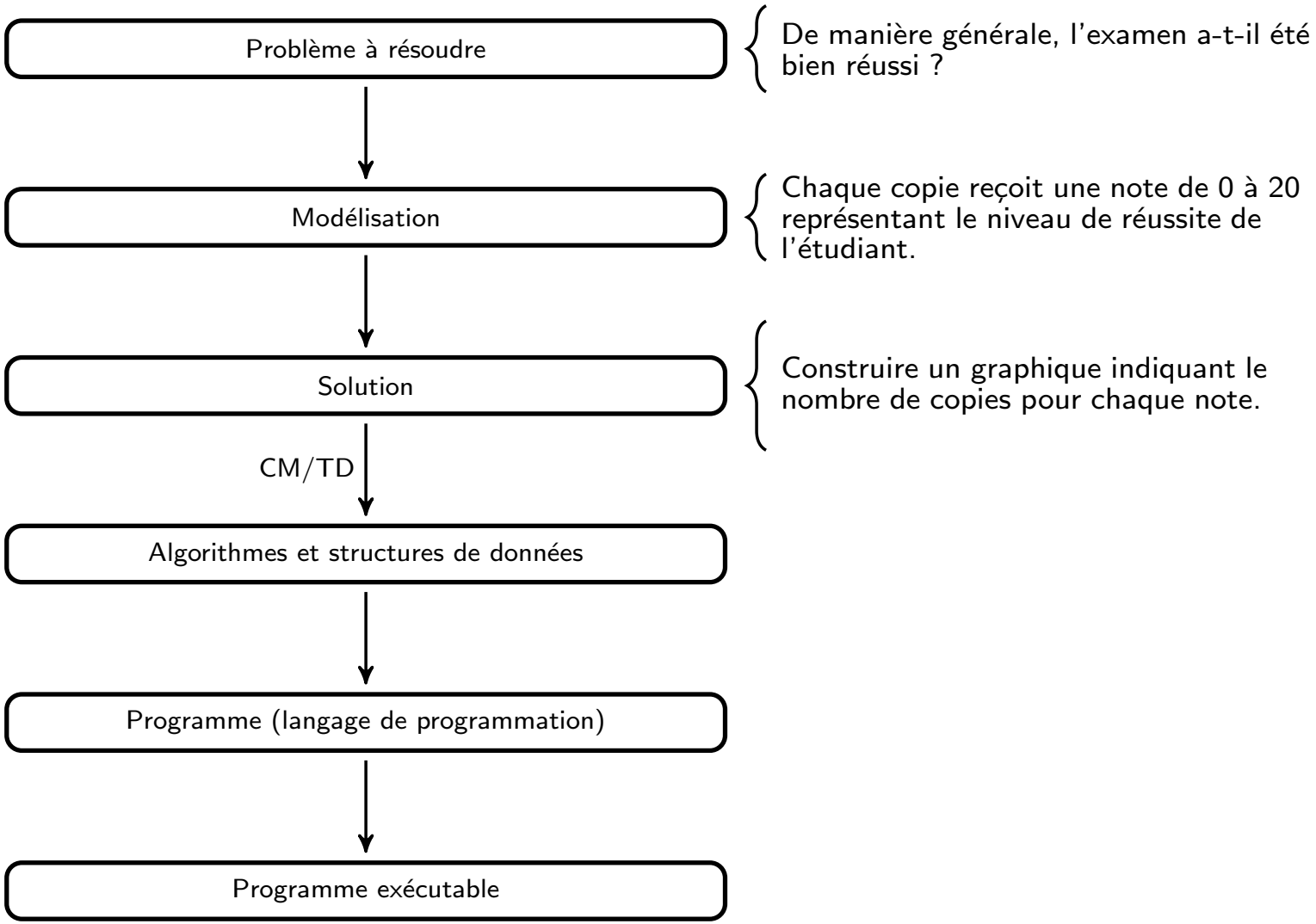
Définir un compteur pour chaque valeur de 0 à 20.
$\longrightarrow$ Pour chaque copie, incrémenter le compteur correspondant à sa note.
Afficher le résultat sous la forme d'un graphique à barres

Definir un tableau à 21 cases.
Initialiser toutes ces cases à 0.
$\longrightarrow$
Pour chaque copie, incrémenter la case correspondante à sa note.
Afficher le résultat sous la forme d'un graphique à barres

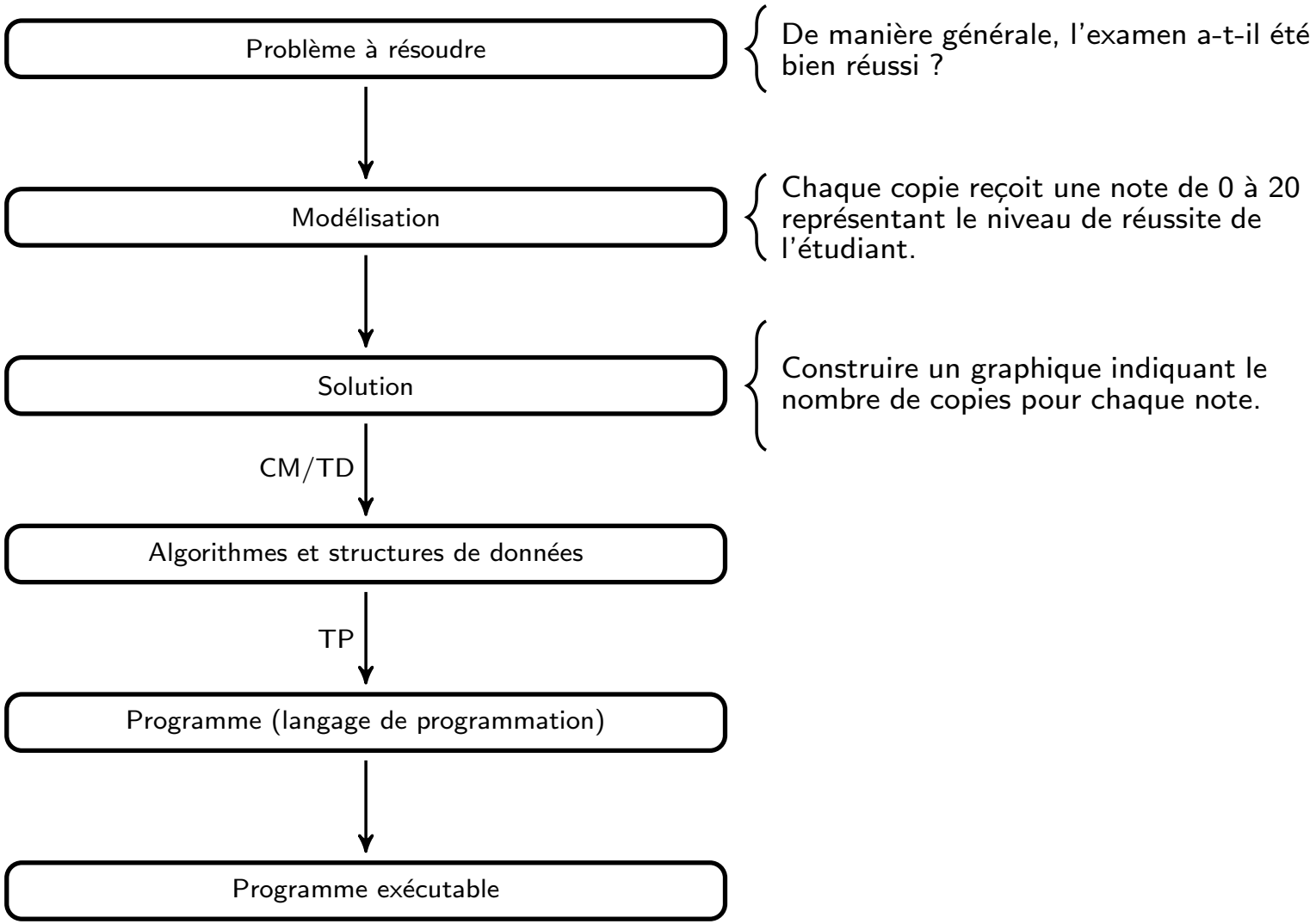
$\vdots$

Début
  t : Tableau d'entier de 0 à 20.
  $n, i, note$: entier
  Pour i **de** 0 à 20 **faire**
    $t[i] := 0$
  fin pour
  Afficher("Entrez le nombre de copies")
$\longrightarrow$ **Lire**(n)
  Pour i **de** 1 à n **faire**
    Afficher("Entrez la note :")
    Lire($note$)
    $T[note] := T[note] + 1$
  fin pour
$\vdots$

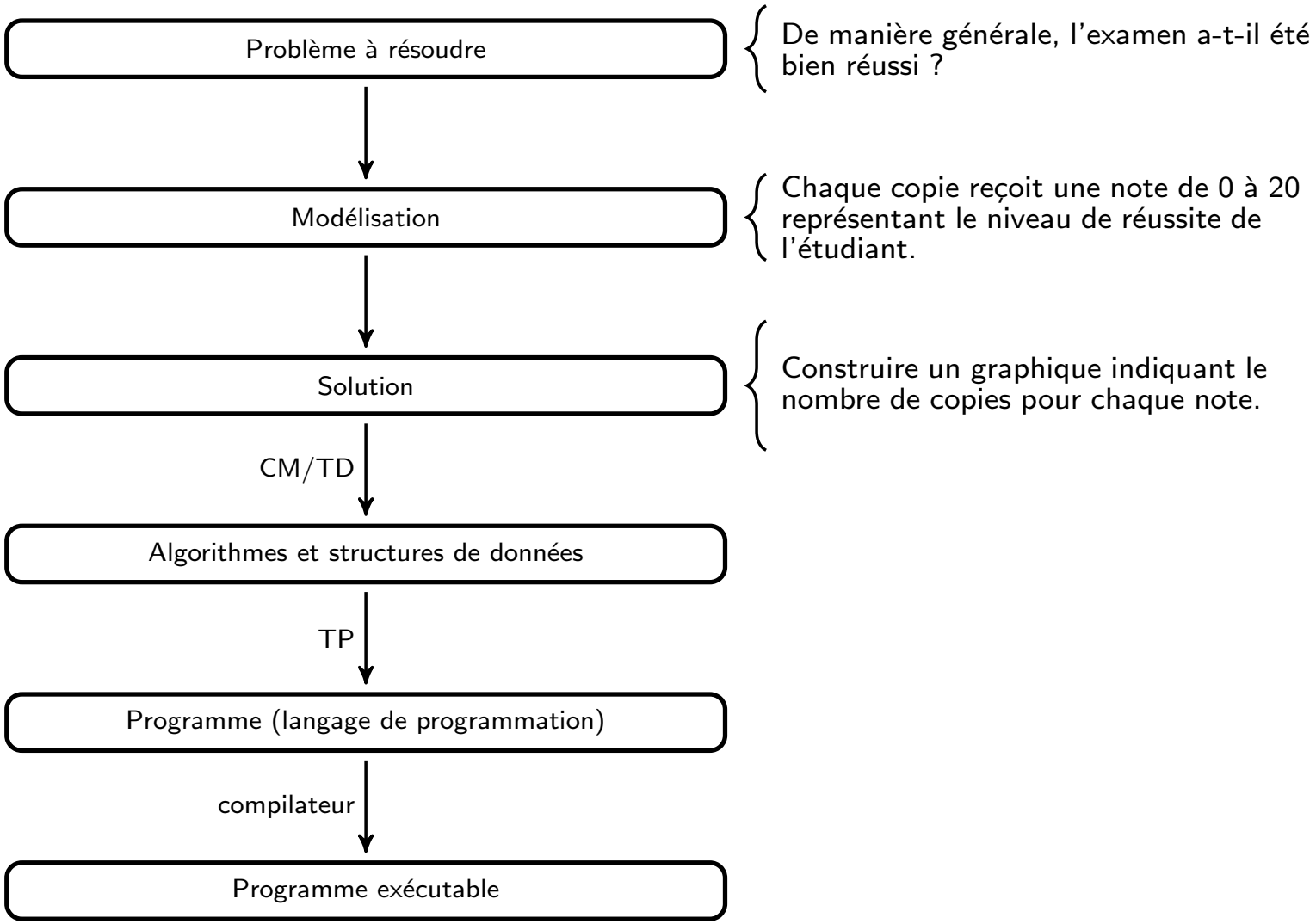
Analyse, algorithmique et programmation



Analyse, algorithmique et programmation



Analyse, algorithmique et programmation



- Langage algorithmique : pseudo-Pascal.

Début
  Affiche("Bonjour le monde")
Fin

- Langage algorithmique : pseudo-Pascal.

```
Début
  Affiche("Bonjour le monde")
Fin
```

- Langage de mise en oeuvre : le langage C.

```
#include <stdio.h>
int main ()
{
  printf("Bonjour le monde\n");
  return 0;
}
```

- Langage algorithmique : pseudo-Pascal.

```
Début
  Affiche("Bonjour le monde")
Fin
```

- Pourquoi utiliser du pseudo-code ?

- Langage de mise en oeuvre : le langage C.

```
#include <stdio.h>
int main ()
{
  printf("Bonjour le monde\n");
  return 0;
}
```

- Pourquoi le langage C ?

- Langage algorithmique : pseudo-Pascal.

```
Début
  Affiche("Bonjour le monde")
Fin

Fonction afficheTab( Tableau t )
  Pour chaque case c du tableau t faire
    Affiche( c )
  Fin pour
Fin
```

- Langage de mise en oeuvre : le langage C.

```
#include <stdio.h>
int main ()
{
  printf("Bonjour le monde\n");
  return 0;
}

void afficheTab( int * t, int n )
{
  int i;
  for ( i=0; i<n; ++i )
  {
    printf( "\%d ", t[i] );
  }
}
```

- Pourquoi utiliser du pseudo-code ?
  - Pour clarifier les idées ! Penser à ce qu'il faut faire AVANT de commencer à coder. Force à se concentrer sur le fond et non sur la forme.
  - Destiné à être lu par des humains, indépendamment de l'environnement de développement (compréhension et communication).
- Pourquoi le langage C ?

- Langage algorithmique : pseudo-Pascal.

```
Début
  Affiche("Bonjour le monde")
Fin

Fonction afficheTab( Tableau t )
  Pour chaque case c du tableau t faire
    Affiche( c )
  Fin pour
Fin
```

- Langage de mise en oeuvre : le langage C.

```
#include <stdio.h>
int main ()
{
  printf("Bonjour le monde\n");
  return 0;
}

void afficheTab( int * t, int n )
{
  int i;
  for ( i=0; i<n; ++i )
  {
    printf( "\%d ", t[i] );
  }
}
```

- Pourquoi utiliser du pseudo-code ?
  - Pour clarifier les idées ! Penser à ce qu'il faut faire AVANT de commencer à coder. Force à se concentrer sur le fond et non sur la forme.
  - Destiné à être lu par des humains, indépendamment de l'environnement de développement (compréhension et communication).
- Pourquoi le langage C ?
  - C est petit (peu de mots-clé).
  - C est commun (abondamment utilisé).
  - C est la base de beaucoup d'autre langages (C++, Java, PHP, AWK, Perl, ...).

Objectifs

- Comprendre et maîtriser les algorithmes de bases sur les tableaux.
  - min/max,
  - argmin/argmax,
  - tris,
  - recherche,
  - mise-à-jour d'un tableau trié,
  - etc.

Objectifs

- Comprendre et maîtriser les algorithmes de bases sur les tableaux.
  - min/max,
  - argmin/argmax,
  - tris,
  - recherche,
  - mise-à-jour d'un tableau trié,
  - etc.
- Comprendre la notion de type abstrait de donnée (TAD) et connaitre les plus courants :
  - file,
  - pile,
  - file de priorité,
  - ensemble,
  - dictionnaire,
  - etc.

Objectifs

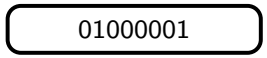
- Comprendre et maîtriser les algorithmes de bases sur les tableaux.
  - min/max,
  - argmin/argmax,
  - tris,
  - recherche,
  - mise-à-jour d'un tableau trié,
  - etc.
- Comprendre la notion de type abstrait de donnée (TAD) et connaitre les plus courants :
  - file,
  - pile,
  - file de priorité,
  - ensemble,
  - dictionnaire,
  - etc.
- Maîtriser leurs utilisations.

Objectifs

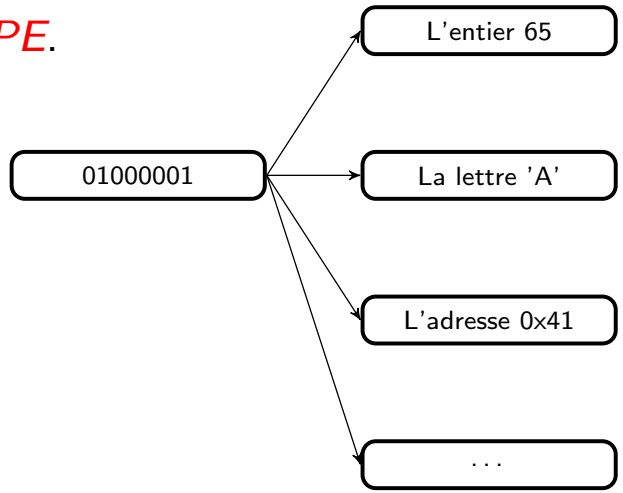
- Comprendre et maîtriser les algorithmes de bases sur les tableaux.
  - min/max,
  - argmin/argmax,
  - tris,
  - recherche,
  - mise-à-jour d'un tableau trié,
  - etc.
- Comprendre la notion de type abstrait de donnée (TAD) et connaitre les plus courants :
  - file,
  - pile,
  - file de priorité,
  - ensemble,
  - dictionnaire,
  - etc.
- Maîtriser leurs utilisations.
- Savoir les mettre en œuvre.

- Variables : désigne une donnée d'un certain *TYPE*.

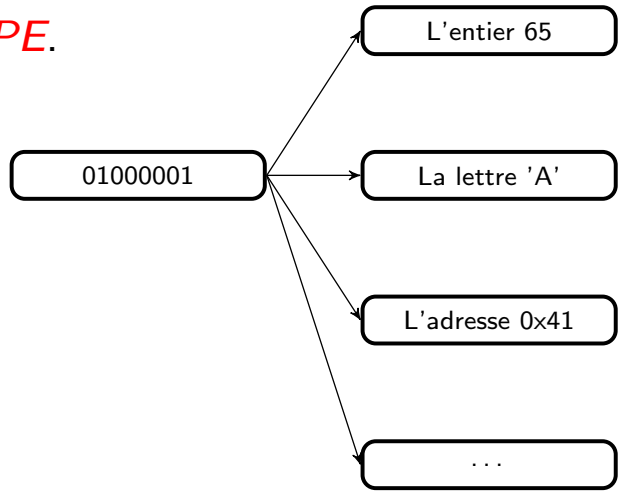
- Variables : désigne une donnée d'un certain *TYPE*.



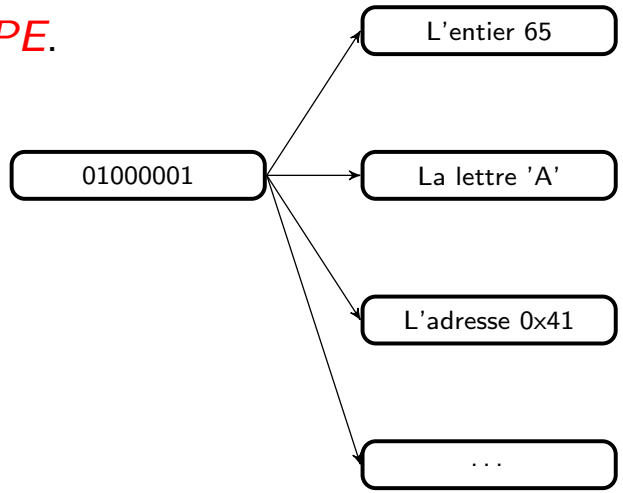
- Variables : désigne une donnée d'un certain *TYPE*.



- Variables : désigne une donnée d'un certain *TYPE*.
- Types de bases :
  - Entier.
  - Flottant (nombre réel, nombre à virgule, ...).
  - Booléen.
  - Caractère.
  - Pointeur (adresse mémoire).



- Variables : désigne une donnée d'un certain *TYPE*.
- Types de bases :
  - Entier.
  - Flottant (nombre réel, nombre à virgule, ...).
  - Booléen.
  - Caractère.
  - Pointeur (adresse mémoire).



- Déclaration des variable.
  - Tout algorithme commence par la *déclaration* de toutes ses variables !
    - Syntaxe : ⟨nom⟩ : ⟨type⟩ initialisé à ⟨valeur initiale⟩.
    - Exemple : **Variables :**
      x : Entier initialisé à -4
      C : caractère initialisé à 'A'
      z : réel initialisé à '3.1417'
      toto : booléen
      titi : Entier

Syntaxe

Variables : ⟨variable 1⟩ : ⟨type⟩ **initialisée à** ⟨valeur initiale⟩
⟨variable 2⟩ : ⟨type⟩ **initialisée à** ⟨valeur initiale⟩
⟨variable 3⟩ : ⟨type⟩ **initialisée à** ⟨valeur initiale⟩
...

Début
⟨instruction⟩
⟨instruction⟩
...
⟨instruction⟩
Fin

- Opérations sur les **Entiers** :

Symbole(s)	Opération	Type de retour
:=	Affectation	–
==, ≠	Égalité/différence	Booléen
>,<,<=,>=	Comparaison	Booléen
+	Addition	Entier
−	Soustraction	Entier
*	Multiplication	Entier
/	Division	Entier
mod	Modulo	Entier

- Opérations sur les **Entiers** :

Symbole(s)	Opération	Type de retour
:=	Affectation	–
==, ≠	Égalité/différence	Booléen
>,<,<=,>=	Comparaison	Booléen
+	Addition	Entier
–	Soustraction	Entier
*	Multiplication	Entier
/	Division	Entier
mod	Modulo	Entier

- Opérations sur les **Flottants** :

Symbole(s)	Opération	Type de retour
:=	Affectation	–
==, ≠	Égalité/différence	Booléen
>,<,<=,>=	Comparaison	Booléen
+	Addition	Flottant
–	Soustraction	Flottant
*	Multiplication	Flottant
/	Division	Flottant

Opérations permises

- Opérations sur les **Booléens** :

Symbole(s)	Opération	Type de retour
:=	Affectation	–
==, ≠	Égalité/différence	Booléen
!	Négation	Booléen
et	Conjonction	Booléen
ou	Disjonction	Booléen
xor	Ou exclusif	Booléen

- Opérations sur les **Caractères** :

Symbole(s)	Opération	Type de retour
:=	Affectation	–
==, ≠	Égalité/différence	Booléen
>,<,<=,>=	Comparaison	Booléen

- **Expression** : une séquence de calcul dont le résultat est une valeur d'un certain type.
  - Exemples :
    - 12
    - $(1 + 2 + 3 + 4 + 5 + 6)/2$
    - $sin(x)$ (où x est un flottant)
    - $(x > 0)$ (où x est un entier)
    - $(x >= \text{'A'})$ ou $(x <= \text{'Z'})$ (où x est un caractère)

- **Expression** : une séquence de calcul dont le résultat est une valeur d'un certain type.
  - Exemples :
    - 12 — type entier
    - $(1 + 2 + 3 + 4 + 5 + 6)/2$
    - $sin(x)$ — (où x est un flottant)
    - $(x > 0)$ — (où x est un entier)
    - $(x >= \text{'A'})$ ou $(x <= \text{'Z'})$ — (où x est un caractère)

- **Expression** : une séquence de calcul dont le résultat est une valeur d'un certain type.
  - Exemples :
    - 12 — type entier
    - $(1 + 2 + 3 + 4 + 5 + 6)/2$ — type entier
    - $sin(x)$ (où x est un flottant)
    - $(x > 0)$ (où x est un entier)
    - $(x >= \text{'A'})$ ou $(x <= \text{'Z'})$ (où x est un caractère)

- **Expression** : une séquence de calcul dont le résultat est une valeur d'un certain type.
  - Exemples :

• 12		type entier
• $(1+2+3+4+5+6)/2$		type entier
• $sin(x)$	(où x est un flottant)	type flottant
• $(x > 0)$	(où x est un entier)	
• $(x >= \text{'A'})$ ou $(x <= \text{'Z'})$	(où x est un caractère)	

- **Expression** : une séquence de calcul dont le résultat est une valeur d'un certain type.
  - Exemples :

• 12		type entier
• $(1+2+3+4+5+6)/2$		type entier
• $sin(x)$	(où x est un flottant)	type flottant
• $(x > 0)$	(où x est un entier)	type booléen
• $(x >= \text{'A'})$ ou $(x <= \text{'Z'})$	(où x est un caractère)	

- **Expression** : une séquence de calcul dont le résultat est une valeur d'un certain type.
  - Exemples :

• 12		type entier
• $(1+2+3+4+5+6)/2$		type entier
• $sin(x)$	(où x est un flottant)	type flottant
• $(x > 0)$	(où x est un entier)	type booléen
• $(x >= \text{'A'})$ ou $(x <= \text{'Z'})$	(où x est un caractère)	type booléen

- **Expression** : une séquence de calcul dont le résultat est une valeur d'un certain type.
  - Exemples :

• 12		type entier
• $(1+2+3+4+5+6)/2$		type entier
• $sin(x)$	(où x est un flottant)	type flottant
• $(x > 0)$	(où x est un entier)	type booléen
• $(x >= \text{'A'})$ ou $(x <= \text{'Z'})$	(où x est un caractère)	type booléen

- Affectation : modifie la valeur d'une variable.
  - Notation : $\langle \text{variable} \rangle := \langle \text{expression} \rangle$ (du même type)
  - Exemples : $x := 10$
    $x := (x + 10) - 5$

- **Expression** : une séquence de calcul dont le résultat est une valeur d'un certain type.
  - Exemples :

• 12		type entier
• $(1+2+3+4+5+6)/2$		type entier
• $sin(x)$	(où x est un flottant)	type flottant
• $(x > 0)$	(où x est un entier)	type booléen
• $(x >= \text{'A'})$ ou $(x <= \text{'Z'})$	(où x est un caractère)	type booléen

- Affectation : modifie la valeur d'une variable.
  - Notation : ⟨variable⟩ := ⟨expression⟩ (du même type)
  - Exemples : $x := 10$
    $x := (x + 10) - 5$

- Entrées sorties : interaction avec le monde extérieur.
  - Lecture au clavier :
    - **Lire**(⟨variable⟩)
  - Affichage à l'écran :
    - **Affiche**(⟨expression⟩)

- Structures de contrôle : influence l'ordre dans lequel sont exécutées les instructions.

si / alors / sinon

Si ⟨ expression booléenne ⟩ **alors**
⟨ instructions à exécuter si la condition est **Vrai**⟩
Sinon
⟨ instructions à exécuter si la condition est **Faux**⟩
Fin si

- Structures de contrôle : influence l'ordre dans lequel sont exécutées les instructions.

si / alors / sinon

Si ⟨ expression booléenne ⟩ **alors**
⟨ instructions à exécuter si la condition est **Vrai**⟩
Sinon
⟨ instructions à exécuter si la condition est **Faux**⟩
Fin si

Variables : n : entier
Début
Affiche("Veuillez entrer un nombre :")
Lire(n)
Si $n < 0$ **alors**
$n := (-n)$
Fin si
Affiche(n)
Fin

- Structures de contrôle : influence l'ordre dans lequel sont exécutées les instructions.

si / alors / sinon

Si ⟨ expression booléenne ⟩ **alors**
⟨ instructions à exécuter si la condition est **Vrai**⟩
Sinon
⟨ instructions à exécuter si la condition est **Faux**⟩
Fin si

Variables : a, b : booléens ...
⟨ ... ⟩
Si ((**non** a) == **Faux**) **alors**
$b :=$ **Vrai**
Sinon
$b :=$ **Faux**
Fin si
⟨ ... ⟩

- Structures de contrôle : influence l'ordre dans lequel sont exécutées les instructions.

si / alors / sinon

Si ⟨ expression booléenne ⟩ **alors**
⟨ instructions à exécuter si la condition est **Vrai**⟩
Sinon
⟨ instructions à exécuter si la condition est **Faux**⟩
Fin si

Variables : a, b : booléens ...
⟨ ... ⟩
Si ((**non** a) == **Faux**) **alors**
$b :=$ **Vrai**
Sinon
$b :=$ **Faux**
Fin si
⟨ ... ⟩

Variables : a, b : booléens ...
⟨ ... ⟩
Si ($a ==$ **Vrai**) **alors**
$b :=$ **Vrai**
Sinon
$b :=$ **Faux**
Fin si
⟨ ... ⟩

- Structures de contrôle : influence l'ordre dans lequel sont exécutées les instructions.

si / alors / sinon

Si ⟨ expression booléenne ⟩ **alors**
⟨ instructions à exécuter si la condition est **Vrai**⟩
Sinon
⟨ instructions à exécuter si la condition est **Faux**⟩
Fin si

Variables : a, b : booléens ...
⟨ ... ⟩
Si ((**non** a) == **Faux**) **alors**
$b :=$ **Vrai**
Sinon
$b :=$ **Faux**
Fin si
⟨ ... ⟩

Variables : a, b : booléens ...
⟨ ... ⟩
Si (a) **alors**
$b :=$ **Vrai**
Sinon
$b :=$ **Faux**
Fin si
⟨ ... ⟩

- Structures de contrôle : influence l'ordre dans lequel sont exécutées les instructions.

si / alors / sinon

Si ⟨ expression booléenne ⟩ **alors**
⟨ instructions à exécuter si la condition est **Vrai**⟩
Sinon
⟨ instructions à exécuter si la condition est **Faux**⟩
Fin si

Variables : a, b : booléens ...
⟨ ... ⟩
Si ((**non** a) == **Faux**) **alors**
$b :=$ **Vrai**
Sinon
$b :=$ **Faux**
Fin si
⟨ ... ⟩

Variables : a, b : booléens ...
⟨ ... ⟩
Si (a) **alors**
$b :=$ **Vrai**
Sinon
$b :=$ **Faux**
Fin si
⟨ ... ⟩

Variables : a, b : booléens ...
⟨ ... ⟩
$b := a$
⟨ ... ⟩

- Structures de contrôles : influence l'ordre dans lequel sont exécuté les instructions.

Boucle "Pour"

Pour ⟨variable⟩ **de** ⟨valeur initiale⟩ **à** ⟨valeur finale⟩ **par pas de** ⟨*increment*⟩ **faire**
  ⟨ instructions ⟩
Fin pour

La section "**par pas de**" est facultative. Par défaut, *increment* vaut 1.

- Structures de contrôles : influence l'ordre dans lequel sont exécuté les instructions.

Boucle "Pour"

Pour ⟨variable⟩ **de** ⟨valeur initiale⟩ **à** ⟨valeur finale⟩ **par pas de** ⟨*increment*⟩ **faire**
⟨ instructions ⟩
Fin pour

La section "**par pas de**" est facultative. Par défaut, *increment* vaut 1.

Variables : n : entier
Début
Affiche("Combien de fois voulez-vous être salué ?")
Lire(n)
Pour i de 1 à n **faire**
Affiche("Bonjour !")
Fin pour
Fin

- Structures de contrôles : influence l'ordre dans lequel sont exécuté les instructions.

Boucle "Tant que"

Tant que ⟨ expression booléenne ⟩ **faire**
⟨ instructions répétées tant que la condition est **Vrai**⟩
Fin tant que

Il est essentiel qu'au moins une des variables intervenant dans l'expression booléenne soit modifiée dans le corps de la boucle !

- Structures de contrôles : influence l'ordre dans lequel sont exécuté les instructions.

Boucle "Tant que"

Tant que ⟨ expression booléenne ⟩ **faire**
⟨ instructions répétées tant que la condition est **Vrai**⟩
Fin tant que

Il est essentiel qu'au moins une des variables intervenant dans l'expression booléenne soit modifiée dans le corps de la boucle !

Variables : x : entier ...
⟨ ... ⟩
Affiche("Veuillez entrer un nombre de 1 et 10")
Lire(x)
Tant que(**non**($1 \leq x$ et $x \leq 10$)) **faire**
  Affiche("Qu'est-ce que vous n'avez pas compris ?!")
  Affiche("Veuillez, encore une fois, entrer un nombre entre 0 et 10")
  Lire(x)
Fin tant que
⟨ ... ⟩

- Structures de contrôles : influence l'ordre dans lequel sont exécuté les instructions.

Boucle "Tant que"

Tant que ⟨ expression booléenne ⟩ **faire**
⟨ instructions répétées tant que la condition est **Vrai**⟩
Fin tant que

Il est essentiel qu'au moins une des variables intervenant dans l'expression booléenne soit modifiée dans le corps de la boucle !

Pour i **de** v_1 **à** v_2 **par pas de** p **faire**
⟨instructions⟩
Fin pour

$\longleftrightarrow$

$i := v\_1$
Tant que $(i \leq v\_2)$ **faire**
⟨instructions⟩
$i := i + p$
Fin tant que

- Structures de contrôles : influence l'ordre dans lequel sont exécuté les instructions.

Boucle "Tant que"

Tant que ⟨ expression booléenne ⟩ **faire**
⟨ instructions répétées tant que la condition est **Vrai**⟩
Fin tant que

Il est essentiel qu'au moins une des variables intervenant dans l'expression booléenne soit modifiée dans le corps de la boucle !

Pour i **de** v_1 **à** v_2 **par pas de** p **faire**
⟨instructions⟩
Fin pour

$\longleftrightarrow$

$i := v\_1$
Tant que $(i \leq v\_2)$ **faire**
⟨instructions⟩
$i := i + p$
Fin tant que

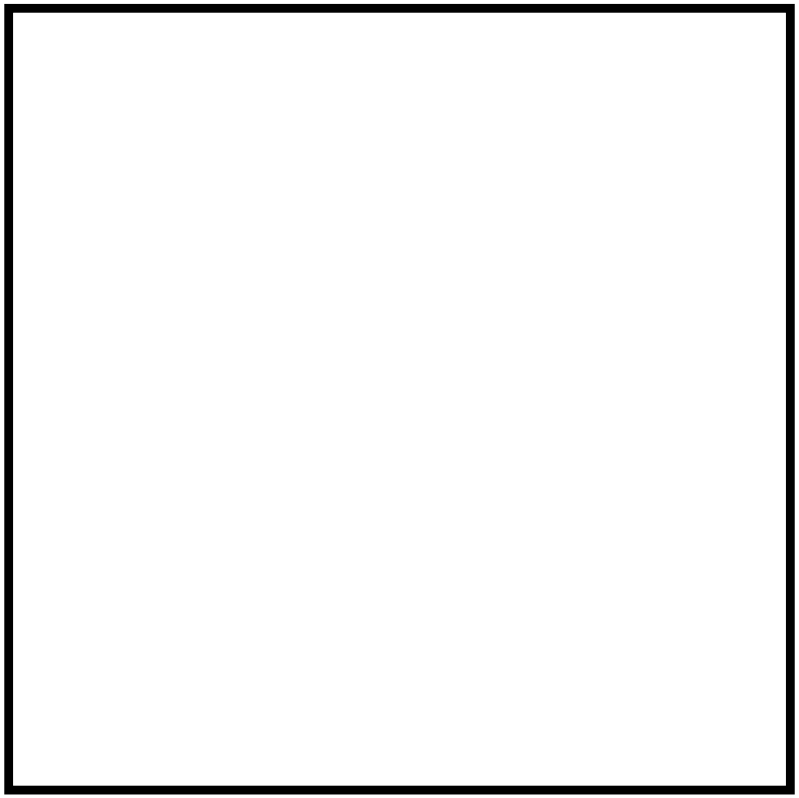
Règle génerale : la boucle "Pour" est utilisée lorsque le nombre exact d'itérations à affectuer est connu AVANT de commencer.

Trace d'exécution

Variables : a : réel initialisé à 0.0
b : réel initialise à 2.0
m : réel

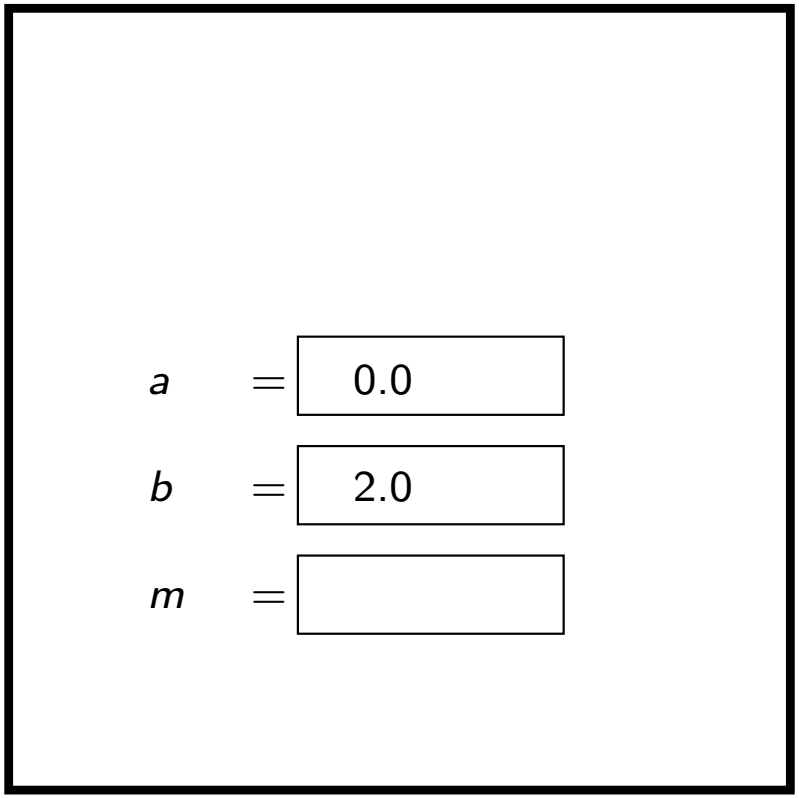
Début

Tant que $b - a > 0.2$ **faire**
$m := (a + b)/2.0$
Si $m * m < 2.0$ **alors**
$a := m$
Sinon
$b := m$
Fin si
Fin tant que
Affiche(a)
Fin



Trace d'exécution

Variables : a : réel initialisé à 0.0
b : réel initialise à 2.0
m : réel
Début
→ **Tant que** $b - a > 0.2$ **faire**
$m := (a + b)/2.0$
Si $m * m < 2.0$ **alors**
$a := m$
Sinon
$b := m$
Fin si
Fin tant que
Affiche(a)
Fin



Trace d'exécution

Variables : a : réel initialisé à 0.0
b : réel initialise à 2.0
m : réel

Début

Tant que $b - a > 0.2$ **faire**

→ $m := (a + b)/2.0$

Si $m * m < 2.0$ **alors**

$a := m$

Sinon

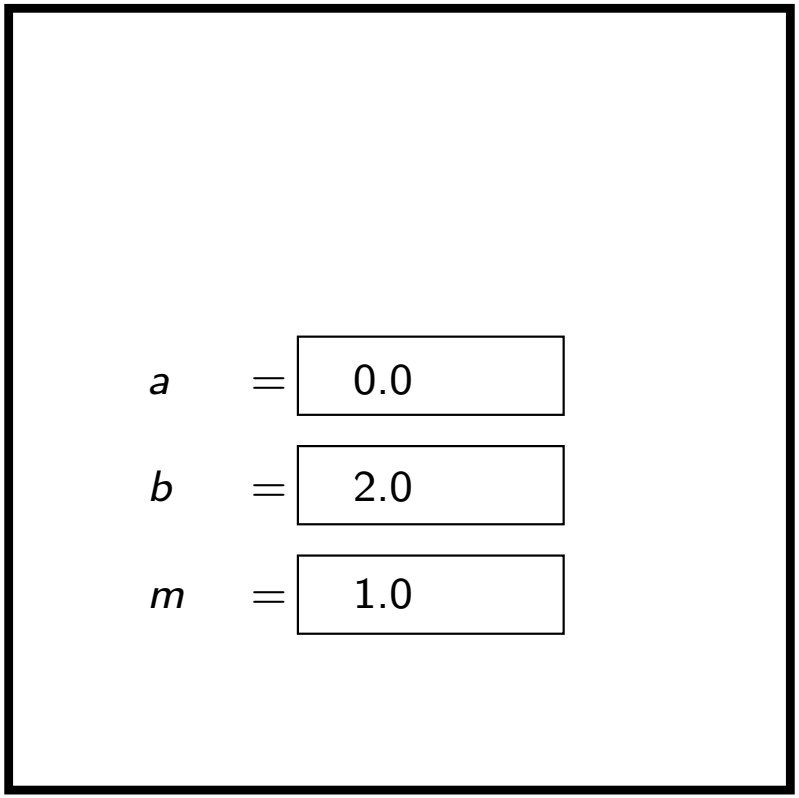
$b := m$

Fin si

Fin tant que

Affiche(a)

Fin

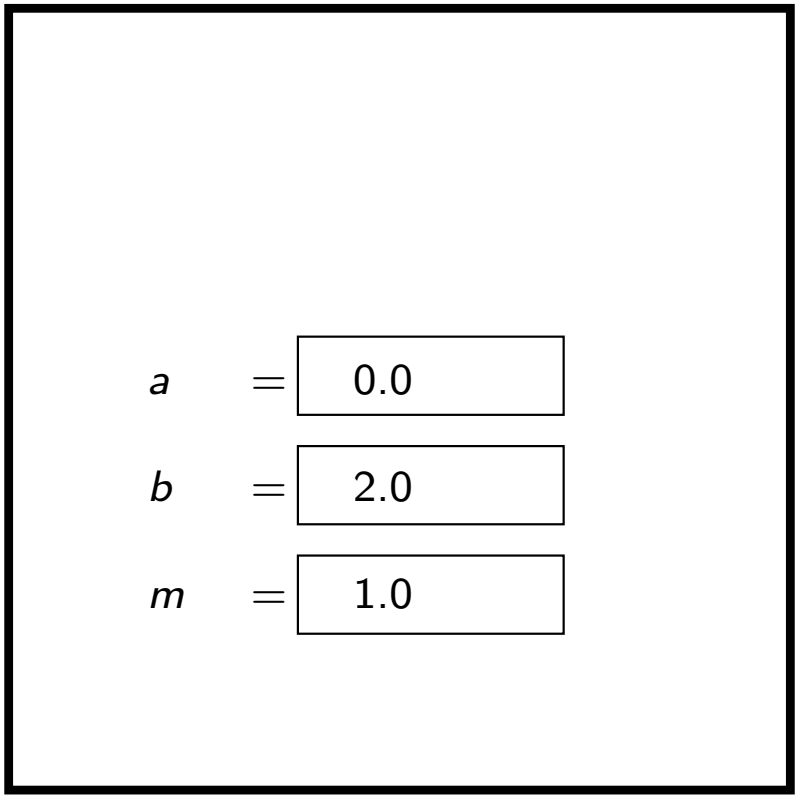


Trace d'exécution

Variables : a : réel initialisé à 0.0
b : réel initialise à 2.0
m : réel

Début

Tant que $b - a > 0.2$ **faire**
$m := (a + b)/2.0$
→ **Si** $m * m < 2.0$ **alors**
$a := m$
Sinon
$b := m$
Fin si
Fin tant que
Affiche(a)
Fin

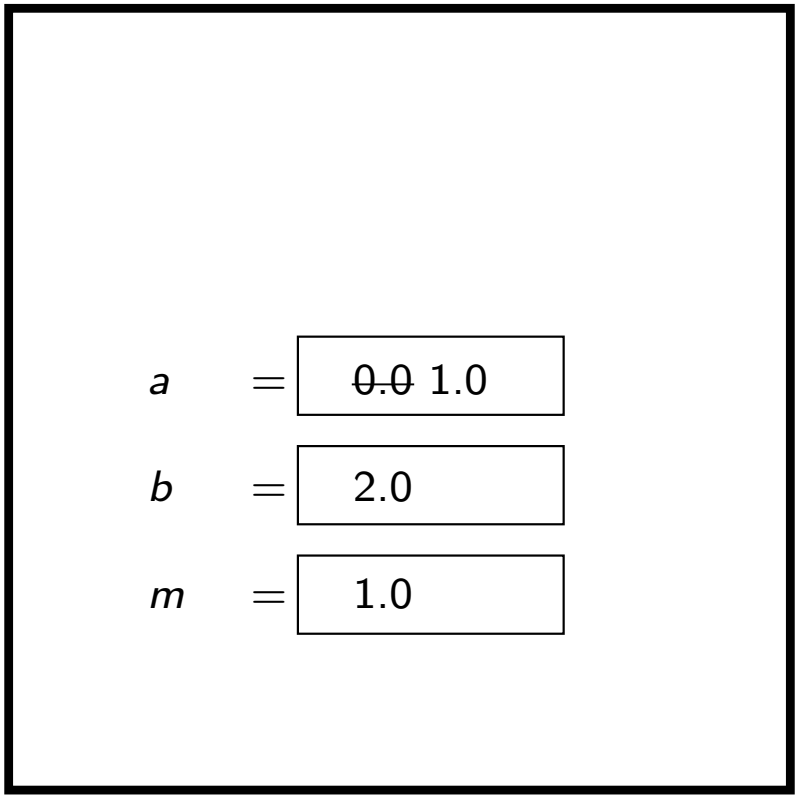


Trace d'exécution

Variables : a : réel initialisé à 0.0
b : réel initialise à 2.0
m : réel

Début

Tant que $b - a > 0.2$ **faire**
$m := (a + b)/2.0$
Si $m * m < 2.0$ **alors**
→ $a := m$
Sinon
$b := m$
Fin si
Fin tant que
Affiche(a)
Fin

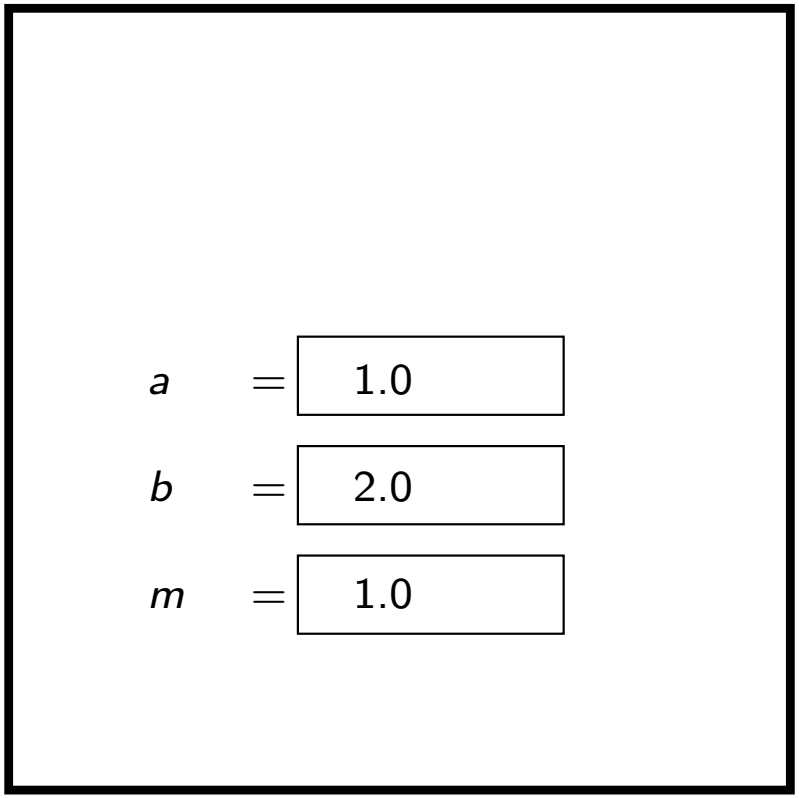


Trace d'exécution

Variables : a : réel initialisé à 0.0
b : réel initialise à 2.0
m : réel

Début

Tant que $b - a > 0.2$ **faire**
$m := (a + b)/2.0$
Si $m * m < 2.0$ **alors**
$a := m$
→ **Sinon**
$b := m$
Fin si
Fin tant que
Affiche(a)
Fin

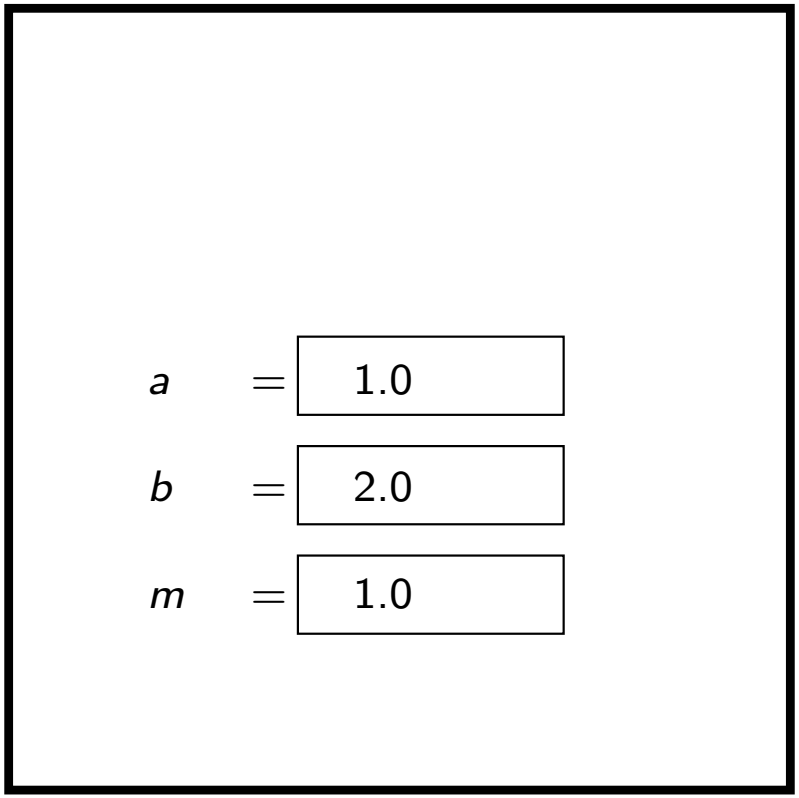


Trace d'exécution

Variables : a : réel initialisé à 0.0
b : réel initialise à 2.0
m : réel

Début

Tant que $b - a > 0.2$ **faire**
$m := (a + b)/2.0$
Si $m * m < 2.0$ **alors**
$a := m$
Sinon
$b := m$
→ **Fin si**
Fin tant que
Affiche(a)
Fin

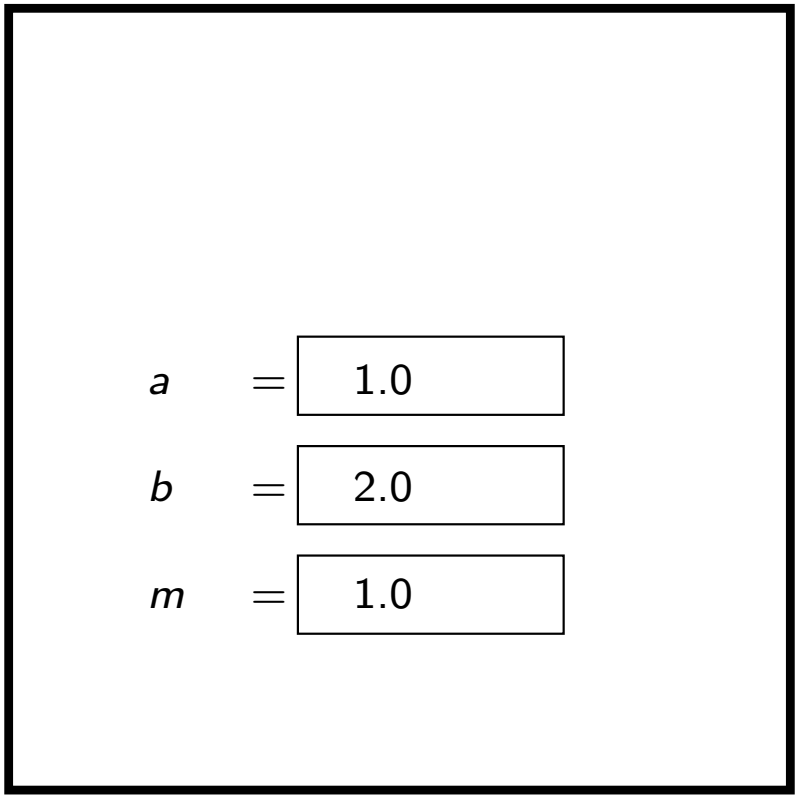


Trace d'exécution

Variables : a : réel initialisé à 0.0
b : réel initialise à 2.0
m : réel

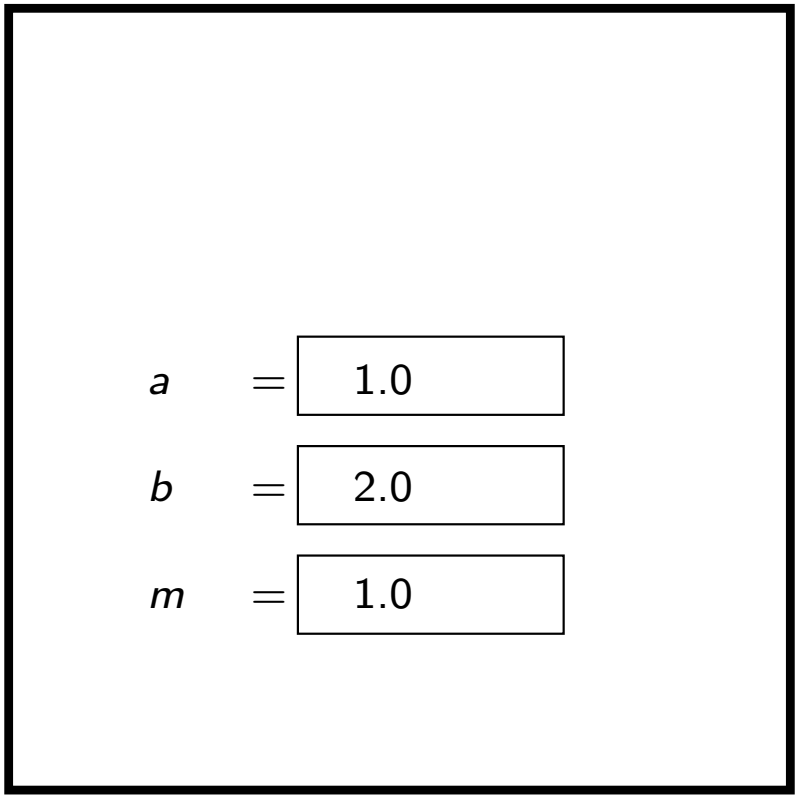
Début

Tant que $b - a > 0.2$ **faire**
$m := (a + b)/2.0$
Si $m * m < 2.0$ **alors**
$a := m$
Sinon
$b := m$
Fin si
→ **Fin tant que**
Affiche(a)
Fin



Trace d'exécution

Variables : a : réel initialisé à 0.0
b : réel initialise à 2.0
m : réel
Début
→ **Tant que** $b - a > 0.2$ **faire**
$m := (a + b)/2.0$
Si $m * m < 2.0$ **alors**
$a := m$
Sinon
$b := m$
Fin si
Fin tant que
Affiche(a)
Fin



Trace d'exécution

Variables : a : réel initialisé à 0.0
b : réel initialise à 2.0
m : réel

Début

Tant que $b - a > 0.2$ **faire**

→ $m := (a + b)/2.0$

Si $m * m < 2.0$ **alors**

$a := m$

Sinon

$b := m$

Fin si

Fin tant que

Affiche(a)

Fin

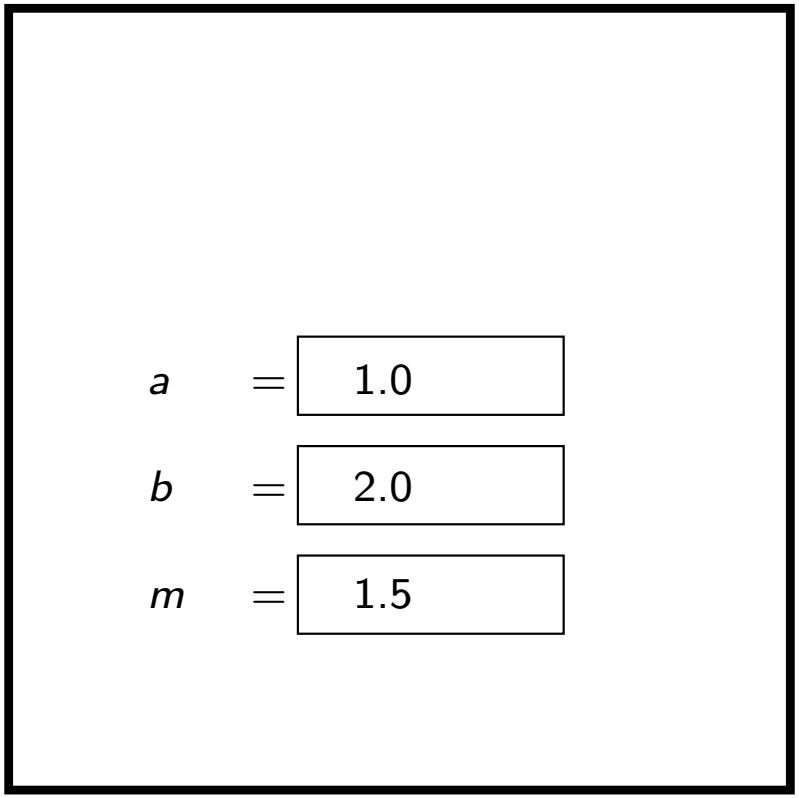
a	=	1.0
b	=	2.0
m	=	~~1.0~~ 1.5

Trace d'exécution

Variables : a : réel initialisé à 0.0
b : réel initialise à 2.0
m : réel

Début

Tant que $b - a > 0.2$ **faire**
$m := (a + b)/2.0$
→ **Si** $m * m < 2.0$ **alors**
$a := m$
Sinon
$b := m$
Fin si
Fin tant que
Affiche(a)
Fin



Trace d'exécution

Variables : a : réel initialisé à 0.0
b : réel initialise à 2.0
m : réel

Début

Tant que $b - a > 0.2$ **faire**
$m := (a + b)/2.0$
Si $m * m < 2.0$ **alors**
$a := m$
Sinon
→ $b := m$
Fin si
Fin tant que
Affiche(a)
Fin

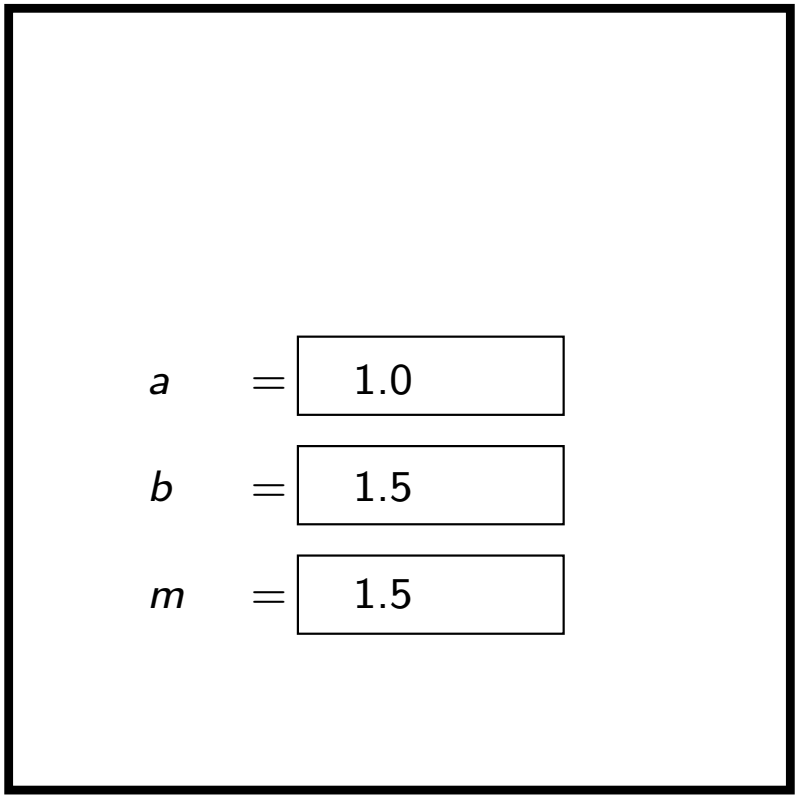
a	=	1.0
b	=	~~2.0~~ 1.5
m	=	1.5

Trace d'exécution

Variables : a : réel initialisé à 0.0
b : réel initialise à 2.0
m : réel

Début

Tant que $b - a > 0.2$ **faire**
$m := (a + b)/2.0$
Si $m * m < 2.0$ **alors**
$a := m$
Sinon
$b := m$
→ **Fin si**
Fin tant que
Affiche(a)
Fin

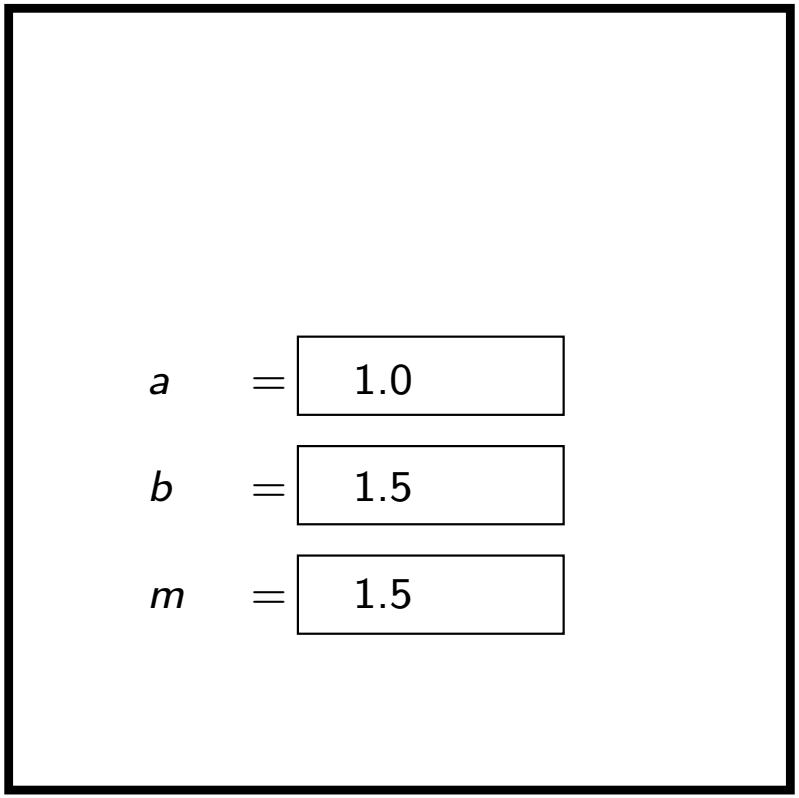


Trace d'exécution

Variables : a : réel initialisé à 0.0
b : réel initialise à 2.0
m : réel

Début

Tant que $b - a > 0.2$ **faire**
$m := (a + b)/2.0$
Si $m * m < 2.0$ **alors**
$a := m$
Sinon
$b := m$
Fin si
→ **Fin tant que**
Affiche(a)
Fin



Trace d'exécution

Variables : a : réel initialisé à 0.0
b : réel initialise à 2.0
m : réel
Début
→ **Tant que** $b - a > 0.2$ **faire**
$m := (a + b)/2.0$
Si $m * m < 2.0$ **alors**
$a := m$
Sinon
$b := m$
Fin si
Fin tant que
Affiche(a)
Fin

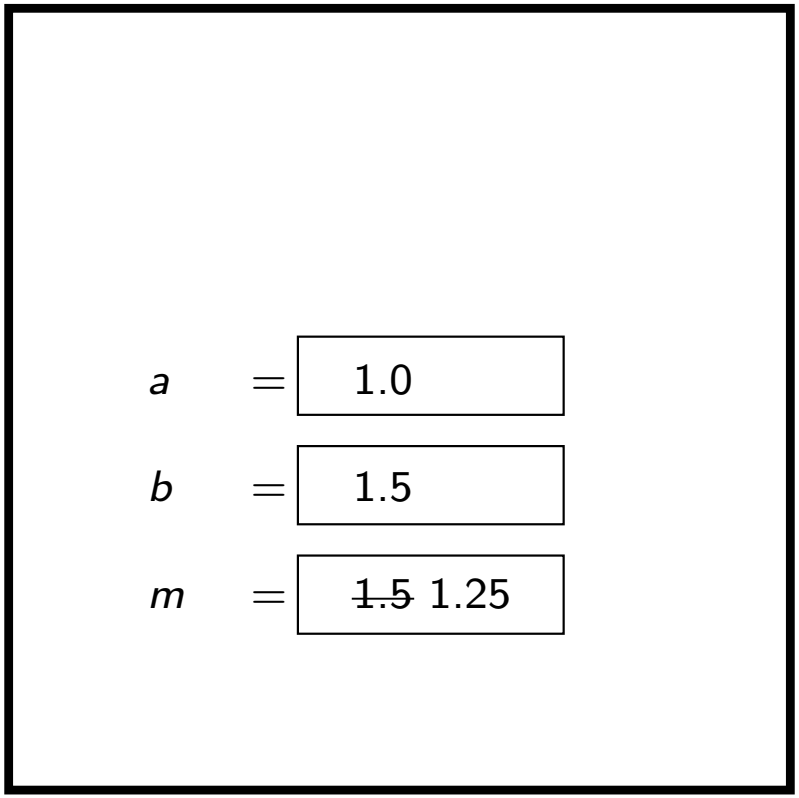
a	=	1.0
b	=	1.5
m	=	1.5

Trace d'exécution

Variables : a : réel initialisé à 0.0
b : réel initialise à 2.0
m : réel

Début

Tant que $b - a > 0.2$ **faire**
→ $m := (a + b)/2.0$
Si $m * m < 2.0$ **alors**
$a := m$
Sinon
$b := m$
Fin si
Fin tant que
Affiche(a)
Fin

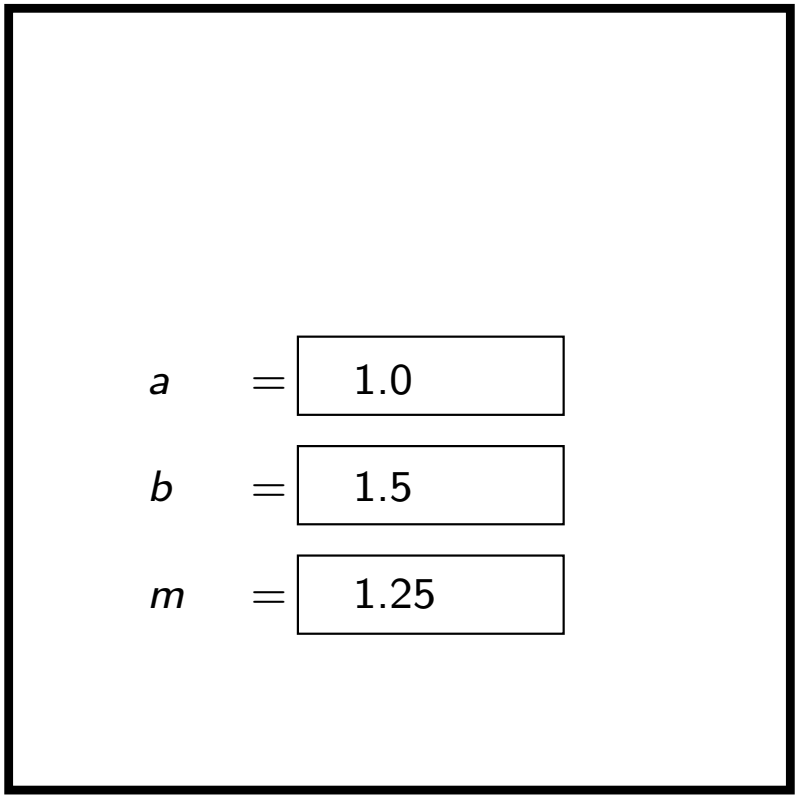


Trace d'exécution

Variables : a : réel initialisé à 0.0
b : réel initialise à 2.0
m : réel

Début

Tant que $b - a > 0.2$ **faire**
$m := (a + b)/2.0$
→ **Si** $m * m < 2.0$ **alors**
$a := m$
Sinon
$b := m$
Fin si
Fin tant que
Affiche(a)
Fin



Trace d'exécution

Variables : a : réel initialisé à 0.0
b : réel initialise à 2.0
m : réel

Début

Tant que $b - a > 0.2$ **faire**
$m := (a + b)/2.0$
Si $m * m < 2.0$ **alors**
→ $a := m$
Sinon
$b := m$
Fin si
Fin tant que
Affiche(a)
Fin

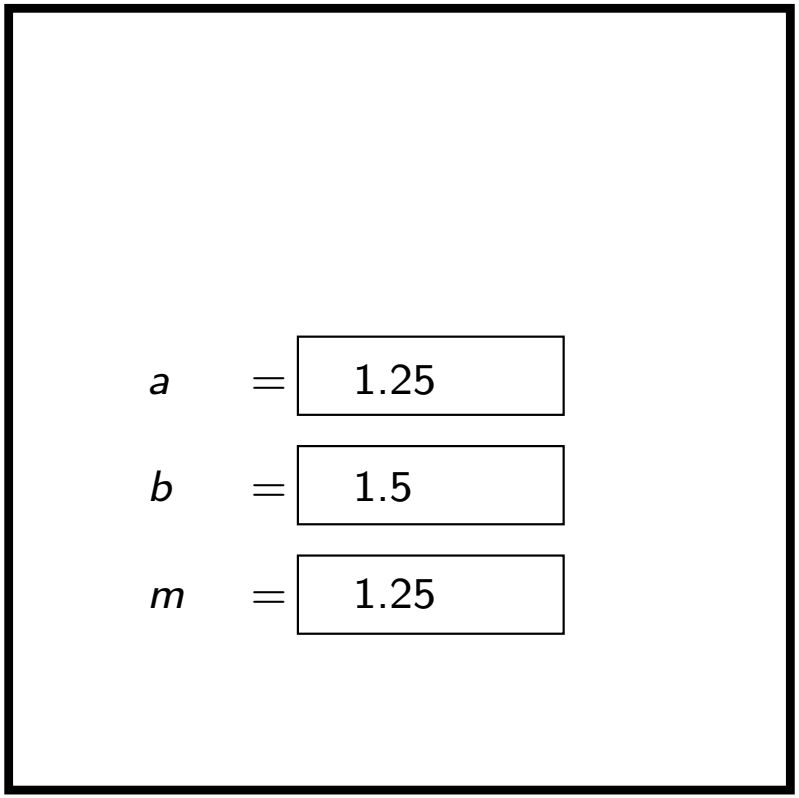
a	=	~~1.0~~ 1.25
b	=	1.5
m	=	1.25

Trace d'exécution

Variables : a : réel initialisé à 0.0
b : réel initialise à 2.0
m : réel

Début

Tant que $b - a > 0.2$ **faire**
$m := (a + b)/2.0$
Si $m * m < 2.0$ **alors**
$a := m$
→ **Sinon**
$b := m$
Fin si
Fin tant que
Affiche(a)
Fin



Trace d'exécution

Variables : a : réel initialisé à 0.0
b : réel initialise à 2.0
m : réel

Début

Tant que $b - a > 0.2$ **faire**
$m := (a + b)/2.0$
Si $m * m < 2.0$ **alors**
$a := m$
Sinon
$b := m$
→ **Fin si**
Fin tant que
Affiche(a)
Fin

a	=	1.25
b	=	1.5
m	=	1.25

Trace d'exécution

Variables : a : réel initialisé à 0.0
b : réel initialise à 2.0
m : réel

Début

  Tant que $b - a > 0.2$ **faire**
    $m := (a + b)/2.0$
    Si $m * m < 2.0$ **alors**
      $a := m$
    Sinon
      $b := m$
    Fin si
→ **Fin tant que**
  Affiche(a)

Fin

a	=	1.25
b	=	1.5
m	=	1.25

Trace d'exécution

Variables : a : réel initialisé à 0.0
b : réel initialise à 2.0
m : réel
Début
→ **Tant que** $b - a > 0.2$ **faire**
$m := (a + b)/2.0$
Si $m * m < 2.0$ **alors**
$a := m$
Sinon
$b := m$
Fin si
Fin tant que
Affiche(a)
Fin

a	=	1.25
b	=	1.5
m	=	1.25

Trace d'exécution

Variables : a : réel initialisé à 0.0
b : réel initialise à 2.0
m : réel

Début

Tant que $b - a > 0.2$ **faire**

→ $m := (a + b)/2.0$

Si $m * m < 2.0$ **alors**

$a := m$

Sinon

$b := m$

Fin si

Fin tant que

Affiche(a)

Fin

a	=	1.25
b	=	1.5
m	=	~~1.25~~ 1.375

Trace d'exécution

Variables : a : réel initialisé à 0.0
b : réel initialise à 2.0
m : réel

Début

  Tant que $b - a > 0.2$ **faire**
    $m := (a + b)/2.0$
→ **Si** $m * m < 2.0$ **alors**
      $a := m$
    Sinon
      $b := m$
    Fin si
  Fin tant que
  Affiche(a)

Fin

a	=	1.25
b	=	1.5
m	=	1.375

Trace d'exécution

Variables : a : réel initialisé à 0.0
b : réel initialise à 2.0
m : réel

Début

Tant que $b - a > 0.2$ **faire**
$m := (a + b)/2.0$
Si $m * m < 2.0$ **alors**
→ $a := m$
Sinon
$b := m$
Fin si
Fin tant que
Affiche(a)
Fin

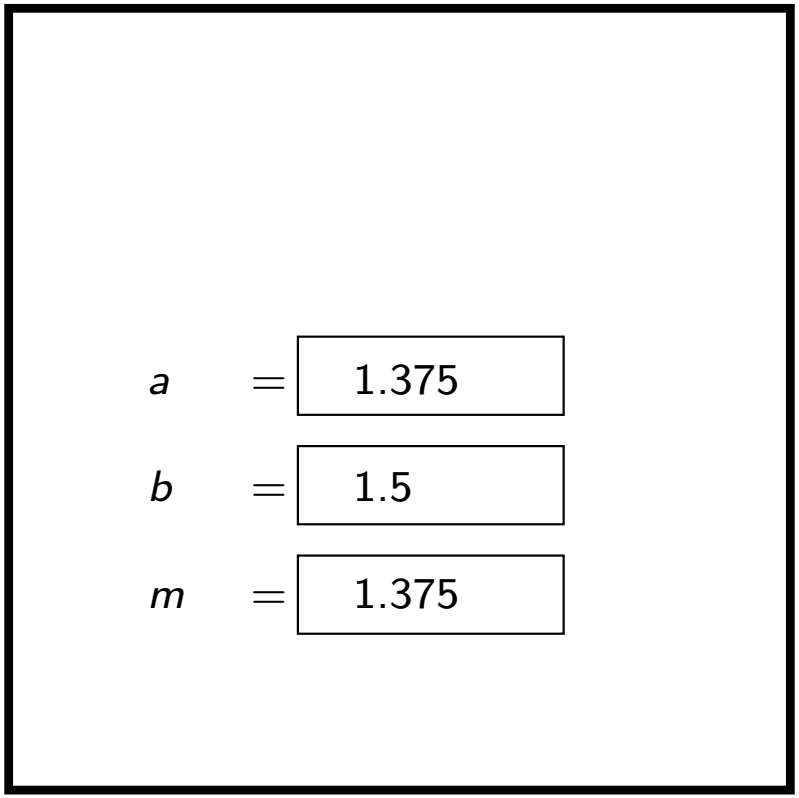
a	=	~~1.25~~ 1.375
b	=	1.5
m	=	1.375

Trace d'exécution

Variables : a : réel initialisé à 0.0
b : réel initialise à 2.0
m : réel

Début

Tant que $b - a > 0.2$ **faire**
$m := (a + b)/2.0$
Si $m * m < 2.0$ **alors**
$a := m$
→ **Sinon**
$b := m$
Fin si
Fin tant que
Affiche(a)
Fin



Trace d'exécution

Variables : a : réel initialisé à 0.0
b : réel initialise à 2.0
m : réel

Début

Tant que $b - a > 0.2$ **faire**
$m := (a + b)/2.0$
Si $m * m < 2.0$ **alors**
$a := m$
Sinon
$b := m$
→ **Fin si**
Fin tant que
Affiche(a)
Fin

a	=	1.375
b	=	1.5
m	=	1.375

Trace d'exécution

Variables : a : réel initialisé à 0.0
b : réel initialise à 2.0
m : réel

Début

Tant que $b - a > 0.2$ **faire**
$m := (a + b)/2.0$
Si $m * m < 2.0$ **alors**
$a := m$
Sinon
$b := m$
Fin si
→ **Fin tant que**
Affiche(a)
Fin

a	=	1.375
b	=	1.5
m	=	1.375

Trace d'exécution

Variables : a : réel initialisé à 0.0
b : réel initialise à 2.0
m : réel
Début
→ **Tant que** $b - a > 0.2$ **faire**
$m := (a + b)/2.0$
Si $m * m < 2.0$ **alors**
$a := m$
Sinon
$b := m$
Fin si
Fin tant que
Affiche(a)
Fin

a	=	1.375
b	=	1.5
m	=	1.375

Trace d'exécution

Variables : a : réel initialisé à 0.0
b : réel initialise à 2.0
m : réel

Début
  Tant que $b - a > 0.2$ **faire**
    $m := (a + b)/2.0$
    Si $m * m < 2.0$ **alors**
      $a := m$
    Sinon
      $b := m$
    Fin si
  Fin tant que
→ **Affiche**(a)
Fin

a	=	1.375
b	=	1.5
m	=	1.375

Tableaux

Contient un nombre fixé d'éléments du même type.

- Les éléments sont rangés consécutivement en mémoire.
- Impossible de modifier le nombre d'éléments (déterminé à l'initialisation).
- Les éléments sont rangés dans des *cases* identifier par des entiers successif. Les cases d'un tableau à n éléments sont habituellement identifiés de 0 à $n-1$.
- On accède aux cases d'un tableau via les crochets [⟨*nocase*⟩].

Déclaration :
⟨variable⟩ : Tableau[⟨*debut*⟩..⟨*fin*⟩] de ⟨ type des éléments ⟩
où *debut* et *fin* sont, respectivement, l'indice de la première et la dernière case.

Tableaux

Contient un nombre fixé d'éléments du même type.

- Les éléments sont rangés consécutivement en mémoire.
- Impossible de modifier le nombre d'éléments (déterminé à l'initialisation).
- Les éléments sont rangés dans des *cases* identifier par des entiers successif. Les cases d'un tableau à n éléments sont habituellement identifiés de 0 à $n-1$.
- On accède aux cases d'un tableau via les crochets [⟨*nocase*⟩].

Déclaration :
⟨variable⟩ : Tableau[⟨*debut*⟩..⟨*fin*⟩] de ⟨ type des éléments ⟩
où *debut* et *fin* sont, respectivement, l'indice de la première et la dernière case.

Exemple de trace

Variables : T : tableau[0..9] d'Entiers
i : entier
Début
Pour i **de** 0 **à** 9 **faire**
$T[i] := i \mod 3$
Fin pour
Fin

Tableaux

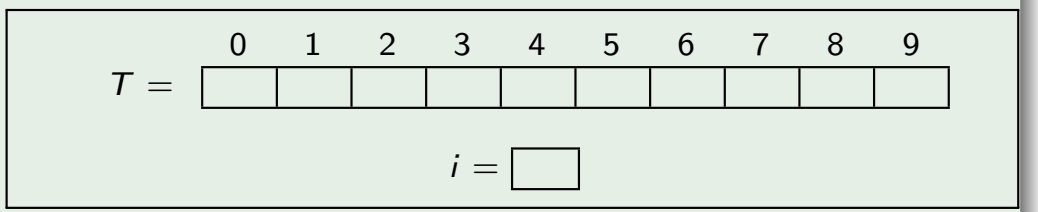
Contient un nombre fixé d'éléments du même type.

- Les éléments sont rangés consécutivement en mémoire.
- Impossible de modifier le nombre d'éléments (déterminé à l'initialisation).
- Les éléments sont rangés dans des *cases* identifier par des entiers successif. Les cases d'un tableau à n éléments sont habituellement identifiés de 0 à $n-1$.
- On accède aux cases d'un tableau via les crochets [⟨*nocase*⟩].

Déclaration :
⟨variable⟩ : Tableau[⟨*debut*⟩..⟨*fin*⟩] de ⟨ type des éléments ⟩
où *debut* et *fin* sont, respectivement, l'indice de la première et la dernière case.

Exemple de trace

Variables : T : tableau[0..9] d'Entiers
i : entier
Début
Pour i **de** 0 **à** 9 **faire**
$T[i] := i \mod 3$
Fin pour
Fin



Tableaux

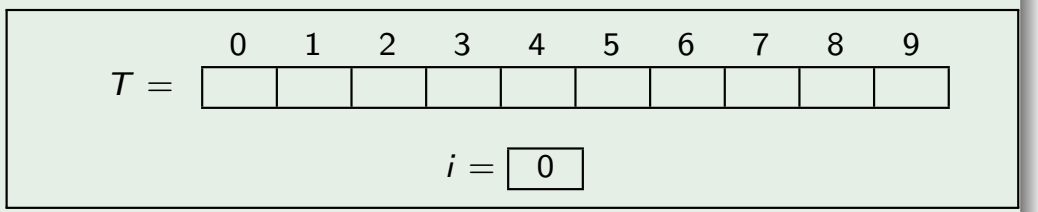
Contient un nombre fixé d'éléments du même type.

- Les éléments sont rangés consécutivement en mémoire.
- Impossible de modifier le nombre d'éléments (déterminé à l'initialisation).
- Les éléments sont rangés dans des *cases* identifier par des entiers successif. Les cases d'un tableau à n éléments sont habituellement identifiés de 0 à $n-1$.
- On accède aux cases d'un tableau via les crochets [⟨*nocase*⟩].

Déclaration :
⟨variable⟩ : Tableau[⟨*debut*⟩..⟨*fin*⟩] de ⟨ type des éléments ⟩
où *debut* et *fin* sont, respectivement, l'indice de la première et la dernière case.

Exemple de trace

```
     Variables :   T : tableau[0..9] d'Entiers
                   i : entier
     Début
→      Pour i de 0 à 9 faire
         T[i] := i  mod 3
       Fin pour
     Fin
```



Tableaux

Contient un nombre fixé d'éléments du même type.

- Les éléments sont rangés consécutivement en mémoire.
- Impossible de modifier le nombre d'éléments (déterminé à l'initialisation).
- Les éléments sont rangés dans des *cases* identifier par des entiers successif. Les cases d'un tableau à n éléments sont habituellement identifiés de 0 à $n-1$.
- On accède aux cases d'un tableau via les crochets [⟨*nocase*⟩].

Déclaration :
⟨variable⟩ : Tableau[⟨*debut*⟩..⟨*fin*⟩] de ⟨ type des éléments ⟩
où *debut* et *fin* sont, respectivement, l'indice de la première et la dernière case.

Exemple de trace

```
    Variables :   T : tableau[0..9] d'Entiers
                  i : entier
    Début
      Pour i de 0 à 9 faire
→       T[i] := i  mod 3
      Fin pour
    Fin
```

	0	1	2	3	4	5	6	7	8	9
$T =$	0									

$i =$ 0

Tableaux

Contient un nombre fixé d'éléments du même type.

- Les éléments sont rangés consécutivement en mémoire.
- Impossible de modifier le nombre d'éléments (déterminé à l'initialisation).
- Les éléments sont rangés dans des *cases* identifier par des entiers successif. Les cases d'un tableau à n éléments sont habituellement identifiés de 0 à $n-1$.
- On accède aux cases d'un tableau via les crochets [⟨*nocase*⟩].

Déclaration :
⟨variable⟩ : Tableau[⟨*debut*⟩..⟨*fin*⟩] de ⟨ type des éléments ⟩
où *debut* et *fin* sont, respectivement, l'indice de la première et la dernière case.

Exemple de trace

Variables : T : tableau[0..9] d'Entiers
i : entier
Début
Pour i **de** 0 **à** 9 **faire**
$T[i] := i \mod 3$
→ **Fin pour**
Fin

$T =$

0	1	2	3	4	5	6	7	8	9
0									

$i =$ 0

Tableaux

Contient un nombre fixé d'éléments du même type.

- Les éléments sont rangés consécutivement en mémoire.
- Impossible de modifier le nombre d'éléments (déterminé à l'initialisation).
- Les éléments sont rangés dans des *cases* identifier par des entiers successif. Les cases d'un tableau à n éléments sont habituellement identifiés de 0 à $n-1$.
- On accède aux cases d'un tableau via les crochets [⟨*nocase*⟩].

Déclaration :
⟨variable⟩ : Tableau[⟨*debut*⟩..⟨*fin*⟩] de ⟨ type des éléments ⟩
où *debut* et *fin* sont, respectivement, l'indice de la première et la dernière case.

Exemple de trace

```
    Variables :   T : tableau[0..9] d'Entiers
                  i : entier
    Début
→     Pour i de 0 à 9 faire
        T[i] := i  mod 3
      Fin pour
    Fin
```

$T =$	0	1	2	3	4	5	6	7	8	9
	0									

$i =$ 1

Tableaux

Contient un nombre fixé d'éléments du même type.

- Les éléments sont rangés consécutivement en mémoire.
- Impossible de modifier le nombre d'éléments (déterminé à l'initialisation).
- Les éléments sont rangés dans des *cases* identifier par des entiers successif. Les cases d'un tableau à n éléments sont habituellement identifiés de 0 à $n-1$.
- On accède aux cases d'un tableau via les crochets [⟨*nocase*⟩].

Déclaration :
⟨variable⟩ : Tableau[⟨*debut*⟩..⟨*fin*⟩] de ⟨ type des éléments ⟩
où *debut* et *fin* sont, respectivement, l'indice de la première et la dernière case.

Exemple de trace

```
   Variables :   T : tableau[0..9] d'Entiers
                 i : entier
   Début
     Pour i de 0 à 9 faire
→      T[i] := i  mod 3
     Fin pour
   Fin
```

	0	1	2	3	4	5	6	7	8	9
$T =$	0	1								

$i =$ 1

Tableaux

Contient un nombre fixé d'éléments du même type.

- Les éléments sont rangés consécutivement en mémoire.
- Impossible de modifier le nombre d'éléments (déterminé à l'initialisation).
- Les éléments sont rangés dans des *cases* identifier par des entiers successif. Les cases d'un tableau à n éléments sont habituellement identifiés de 0 à $n-1$.
- On accède aux cases d'un tableau via les crochets [⟨*nocase*⟩].

Déclaration :
⟨variable⟩ : Tableau[⟨*debut*⟩..⟨*fin*⟩] de ⟨ type des éléments ⟩
où *debut* et *fin* sont, respectivement, l'indice de la première et la dernière case.

Exemple de trace

```
    Variables :   T : tableau[0..9] d'Entiers
                  i : entier
    Début
      Pour i de 0 à 9 faire
        T[i] := i  mod 3
→     Fin pour
    Fin
```

	0	1	2	3	4	5	6	7	8	9
$T =$	0	1								

$i =$ 1

Tableaux

Contient un nombre fixé d'éléments du même type.

- Les éléments sont rangés consécutivement en mémoire.
- Impossible de modifier le nombre d'éléments (déterminé à l'initialisation).
- Les éléments sont rangés dans des *cases* identifier par des entiers successif. Les cases d'un tableau à n éléments sont habituellement identifiés de 0 à $n-1$.
- On accède aux cases d'un tableau via les crochets [⟨*nocase*⟩].

Déclaration :
⟨variable⟩ : Tableau[⟨*debut*⟩..⟨*fin*⟩] de ⟨ type des éléments ⟩
où *debut* et *fin* sont, respectivement, l'indice de la première et la dernière case.

Exemple de trace

```
    Variables :   T : tableau[0..9] d'Entiers
                  i : entier
    Début
→     Pour i de 0 à 9 faire
        T[i] := i  mod 3
      Fin pour
    Fin
```

	0	1	2	3	4	5	6	7	8	9
$T =$	0	1								

$i =$ 2

Tableaux

Contient un nombre fixé d'éléments du même type.

- Les éléments sont rangés consécutivement en mémoire.
- Impossible de modifier le nombre d'éléments (déterminé à l'initialisation).
- Les éléments sont rangés dans des *cases* identifier par des entiers successif. Les cases d'un tableau à n éléments sont habituellement identifiés de 0 à $n-1$.
- On accède aux cases d'un tableau via les crochets [⟨*nocase*⟩].

Déclaration :
⟨variable⟩ : Tableau[⟨*debut*⟩..⟨*fin*⟩] de ⟨ type des éléments ⟩
où *debut* et *fin* sont, respectivement, l'indice de la première et la dernière case.

Exemple de trace

```
    Variables :   T : tableau[0..9] d'Entiers
                  i : entier
    Début
      Pour i de 0 à 9 faire
→       T[i] := i  mod 3
      Fin pour
    Fin
```

	0	1	2	3	4	5	6	7	8	9
$T =$	0	1	2							

$i =$ 2

Tableaux

Contient un nombre fixé d'éléments du même type.

- Les éléments sont rangés consécutivement en mémoire.
- Impossible de modifier le nombre d'éléments (déterminé à l'initialisation).
- Les éléments sont rangés dans des *cases* identifier par des entiers successif. Les cases d'un tableau à n éléments sont habituellement identifiés de 0 à $n-1$.
- On accède aux cases d'un tableau via les crochets [⟨*nocase*⟩].

Déclaration :
⟨variable⟩ : Tableau[⟨*debut*⟩..⟨*fin*⟩] de ⟨ type des éléments ⟩
où *debut* et *fin* sont, respectivement, l'indice de la première et la dernière case.

Exemple de trace

```
    Variables :   T : tableau[0..9] d'Entiers
                  i : entier
    Début
      Pour i de 0 à 9 faire
        T[i] := i  mod 3
→     Fin pour
    Fin
```

	0	1	2	3	4	5	6	7	8	9
$T =$	0	1	2							

$i =$ 2

Tableaux

Contient un nombre fixé d'éléments du même type.

- Les éléments sont rangés consécutivement en mémoire.
- Impossible de modifier le nombre d'éléments (déterminé à l'initialisation).
- Les éléments sont rangés dans des *cases* identifier par des entiers successif. Les cases d'un tableau à n éléments sont habituellement identifiés de 0 à $n-1$.
- On accède aux cases d'un tableau via les crochets [⟨*nocase*⟩].

Déclaration :
⟨variable⟩ : Tableau[⟨*debut*⟩..⟨*fin*⟩] de ⟨ type des éléments ⟩
où *debut* et *fin* sont, respectivement, l'indice de la première et la dernière case.

Exemple de trace

```
     Variables :   T : tableau[0..9] d'Entiers
                   i : entier
     Début
→      Pour i de 0 à 9 faire
         T[i] := i  mod 3
       Fin pour
     Fin
```

$T =$

0	1	2	3	4	5	6	7	8	9
0	1	2							

$i =$ 3

Tableaux

Contient un nombre fixé d'éléments du même type.

- Les éléments sont rangés consécutivement en mémoire.
- Impossible de modifier le nombre d'éléments (déterminé à l'initialisation).
- Les éléments sont rangés dans des *cases* identifier par des entiers successif. Les cases d'un tableau à n éléments sont habituellement identifiés de 0 à $n-1$.
- On accède aux cases d'un tableau via les crochets [⟨*nocase*⟩].

Déclaration :
⟨variable⟩ : Tableau[⟨*debut*⟩..⟨*fin*⟩] de ⟨ type des éléments ⟩
où *debut* et *fin* sont, respectivement, l'indice de la première et la dernière case.

Exemple de trace

```
    Variables :   T : tableau[0..9] d'Entiers
                  i : entier
    Début
      Pour i de 0 à 9 faire
→       T[i] := i  mod 3
      Fin pour
    Fin
```

	0	1	2	3	4	5	6	7	8	9
$T =$	0	1	2	0						

$i =$ 3

Tableaux

Contient un nombre fixé d'éléments du même type.

- Les éléments sont rangés consécutivement en mémoire.
- Impossible de modifier le nombre d'éléments (déterminé à l'initialisation).
- Les éléments sont rangés dans des *cases* identifier par des entiers successif. Les cases d'un tableau à n éléments sont habituellement identifiés de 0 à $n-1$.
- On accède aux cases d'un tableau via les crochets [⟨*nocase*⟩].

Déclaration :
⟨variable⟩ : Tableau[⟨*debut*⟩..⟨*fin*⟩] de ⟨ type des éléments ⟩
où *debut* et *fin* sont, respectivement, l'indice de la première et la dernière case.

Exemple de trace

```
    Variables :   T : tableau[0..9] d'Entiers
                  i : entier
    Début
      Pour i de 0 à 9 faire
        T[i] := i  mod 3
→     Fin pour
    Fin
```

	0	1	2	3	4	5	6	7	8	9
$T =$	0	1	2	0						

$i =$ 3

Tableaux

Contient un nombre fixé d'éléments du même type.

- Les éléments sont rangés consécutivement en mémoire.
- Impossible de modifier le nombre d'éléments (déterminé à l'initialisation).
- Les éléments sont rangés dans des *cases* identifier par des entiers successif. Les cases d'un tableau à n éléments sont habituellement identifiés de 0 à $n-1$.
- On accède aux cases d'un tableau via les crochets [⟨*nocase*⟩].

Déclaration :
⟨variable⟩ : Tableau[⟨*debut*⟩..⟨*fin*⟩] de ⟨ type des éléments ⟩
où *debut* et *fin* sont, respectivement, l'indice de la première et la dernière case.

Exemple de trace

```
     Variables :   T : tableau[0..9] d'Entiers
                   i : entier
     Début
→      Pour i de 0 à 9 faire
         T[i] := i  mod 3
       Fin pour
     Fin
```

$T =$

0	1	2	3	4	5	6	7	8	9
0	1	2	0						

$i =$ 4

Tableaux

Contient un nombre fixé d'éléments du même type.

- Les éléments sont rangés consécutivement en mémoire.
- Impossible de modifier le nombre d'éléments (déterminé à l'initialisation).
- Les éléments sont rangés dans des *cases* identifier par des entiers successif. Les cases d'un tableau à n éléments sont habituellement identifiés de 0 à $n-1$.
- On accède aux cases d'un tableau via les crochets [⟨*nocase*⟩].

Déclaration :
⟨variable⟩ : Tableau[⟨*debut*⟩..⟨*fin*⟩] de ⟨ type des éléments ⟩
où *debut* et *fin* sont, respectivement, l'indice de la première et la dernière case.

Exemple de trace

```
    Variables :   T : tableau[0..9] d'Entiers
                  i : entier
    Début
      Pour i de 0 à 9 faire
→       T[i] := i  mod 3
      Fin pour
    Fin
```

	0	1	2	3	4	5	6	7	8	9
$T =$	0	1	2	0	1					

$i =$ 4

Tableaux

Contient un nombre fixé d'éléments du même type.

- Les éléments sont rangés consécutivement en mémoire.
- Impossible de modifier le nombre d'éléments (déterminé à l'initialisation).
- Les éléments sont rangés dans des *cases* identifier par des entiers successif. Les cases d'un tableau à n éléments sont habituellement identifiés de 0 à $n-1$.
- On accède aux cases d'un tableau via les crochets [⟨*nocase*⟩].

Déclaration :
⟨variable⟩ : Tableau[⟨*debut*⟩..⟨*fin*⟩] de ⟨ type des éléments ⟩
où *debut* et *fin* sont, respectivement, l'indice de la première et la dernière case.

Exemple de trace

```
    Variables :   T : tableau[0..9] d'Entiers
                  i : entier
    Début
      Pour i de 0 à 9 faire
        T[i] := i  mod 3
→     Fin pour
    Fin
```

	0	1	2	3	4	5	6	7	8	9
$T =$	0	1	2	0	1					

$i =$ 4

Tableaux

Contient un nombre fixé d'éléments du même type.

- Les éléments sont rangés consécutivement en mémoire.
- Impossible de modifier le nombre d'éléments (déterminé à l'initialisation).
- Les éléments sont rangés dans des *cases* identifier par des entiers successif. Les cases d'un tableau à n éléments sont habituellement identifiés de 0 à $n-1$.
- On accède aux cases d'un tableau via les crochets [⟨*nocase*⟩].

Déclaration :
⟨variable⟩ : Tableau[⟨*debut*⟩..⟨*fin*⟩] de ⟨ type des éléments ⟩
où *debut* et *fin* sont, respectivement, l'indice de la première et la dernière case.

Exemple de trace

```
    Variables :   T : tableau[0..9] d'Entiers
                  i : entier
    Début
→     Pour i de 0 à 9 faire
        T[i] := i  mod 3
      Fin pour
    Fin
```

	0	1	2	3	4	5	6	7	8	9
$T =$	0	1	2	0	1					

$i =$ 5

Tableaux

Contient un nombre fixé d'éléments du même type.

- Les éléments sont rangés consécutivement en mémoire.
- Impossible de modifier le nombre d'éléments (déterminé à l'initialisation).
- Les éléments sont rangés dans des *cases* identifier par des entiers successif. Les cases d'un tableau à n éléments sont habituellement identifiés de 0 à $n-1$.
- On accède aux cases d'un tableau via les crochets [⟨*nocase*⟩].

Déclaration :
⟨variable⟩ : Tableau[⟨*debut*⟩..⟨*fin*⟩] de ⟨ type des éléments ⟩
où *debut* et *fin* sont, respectivement, l'indice de la première et la dernière case.

Exemple de trace

```
    Variables :   T : tableau[0..9] d'Entiers
                  i : entier
    Début
      Pour i de 0 à 9 faire
→       T[i] := i  mod 3
      Fin pour
    Fin
```

	0	1	2	3	4	5	6	7	8	9
$T =$	0	1	2	0	1	2				

$i =$ 5

Tableaux

Contient un nombre fixé d'éléments du même type.

- Les éléments sont rangés consécutivement en mémoire.
- Impossible de modifier le nombre d'éléments (déterminé à l'initialisation).
- Les éléments sont rangés dans des *cases* identifier par des entiers successif. Les cases d'un tableau à n éléments sont habituellement identifiés de 0 à $n-1$.
- On accède aux cases d'un tableau via les crochets [⟨*nocase*⟩].

Déclaration :
⟨variable⟩ : Tableau[⟨*debut*⟩..⟨*fin*⟩] de ⟨ type des éléments ⟩
où *debut* et *fin* sont, respectivement, l'indice de la première et la dernière case.

Exemple de trace

```
    Variables :   T : tableau[0..9] d'Entiers
                  i : entier
    Début
      Pour i de 0 à 9 faire
        T[i] := i  mod 3
→     Fin pour
    Fin
```

	0	1	2	3	4	5	6	7	8	9
$T =$	0	1	2	0	1	2				

$i =$ 5

Tableaux

Contient un nombre fixé d'éléments du même type.

- Les éléments sont rangés consécutivement en mémoire.
- Impossible de modifier le nombre d'éléments (déterminé à l'initialisation).
- Les éléments sont rangés dans des *cases* identifier par des entiers successif. Les cases d'un tableau à n éléments sont habituellement identifiés de 0 à $n-1$.
- On accède aux cases d'un tableau via les crochets [⟨*nocase*⟩].

Déclaration :
⟨variable⟩ : Tableau[⟨*debut*⟩..⟨*fin*⟩] de ⟨ type des éléments ⟩
où *debut* et *fin* sont, respectivement, l'indice de la première et la dernière case.

Exemple de trace

```
    Variables :   T : tableau[0..9] d'Entiers
                  i : entier
    Début
→     Pour i de 0 à 9 faire
        T[i] := i  mod 3
      Fin pour
    Fin
```

	0	1	2	3	4	5	6	7	8	9
$T =$	0	1	2	0	1	2				

$i =$ 6

Tableaux

Contient un nombre fixé d'éléments du même type.

- Les éléments sont rangés consécutivement en mémoire.
- Impossible de modifier le nombre d'éléments (déterminé à l'initialisation).
- Les éléments sont rangés dans des *cases* identifier par des entiers successif. Les cases d'un tableau à n éléments sont habituellement identifiés de 0 à $n-1$.
- On accède aux cases d'un tableau via les crochets [⟨*nocase*⟩].

Déclaration :
⟨variable⟩ : Tableau[⟨*debut*⟩..⟨*fin*⟩] de ⟨ type des éléments ⟩
où *debut* et *fin* sont, respectivement, l'indice de la première et la dernière case.

Exemple de trace

```
    Variables :   T : tableau[0..9] d'Entiers
                  i : entier
    Début
      Pour i de 0 à 9 faire
→       T[i] := i  mod 3
      Fin pour
    Fin
```

	0	1	2	3	4	5	6	7	8	9
$T =$	0	1	2	0	1	2	0			

$i =$ 6

Tableaux

Contient un nombre fixé d'éléments du même type.

- Les éléments sont rangés consécutivement en mémoire.
- Impossible de modifier le nombre d'éléments (déterminé à l'initialisation).
- Les éléments sont rangés dans des *cases* identifier par des entiers successif. Les cases d'un tableau à n éléments sont habituellement identifiés de 0 à $n-1$.
- On accède aux cases d'un tableau via les crochets [⟨*nocase*⟩].

Déclaration :
⟨variable⟩ : Tableau[⟨*debut*⟩..⟨*fin*⟩] de ⟨ type des éléments ⟩
où *debut* et *fin* sont, respectivement, l'indice de la première et la dernière case.

Exemple de trace

```
    Variables :   T : tableau[0..9] d'Entiers
                  i : entier
    Début
      Pour i de 0 à 9 faire
        T[i] := i  mod 3
→     Fin pour
    Fin
```

	0	1	2	3	4	5	6	7	8	9
$T =$	0	1	2	0	1	2	0			

$i =$ 6

Tableaux

Contient un nombre fixé d'éléments du même type.

- Les éléments sont rangés consécutivement en mémoire.
- Impossible de modifier le nombre d'éléments (déterminé à l'initialisation).
- Les éléments sont rangés dans des *cases* identifier par des entiers successif. Les cases d'un tableau à n éléments sont habituellement identifiés de 0 à $n-1$.
- On accède aux cases d'un tableau via les crochets [⟨*nocase*⟩].

Déclaration :
⟨variable⟩ : Tableau[⟨*debut*⟩..⟨*fin*⟩] de ⟨ type des éléments ⟩
où *debut* et *fin* sont, respectivement, l'indice de la première et la dernière case.

Exemple de trace

```
    Variables :   T : tableau[0..9] d'Entiers
                  i : entier
    Début
→     Pour i de 0 à 9 faire
        T[i] := i  mod 3
      Fin pour
    Fin
```

	0	1	2	3	4	5	6	7	8	9
$T =$	0	1	2	0	1	2	0			

$i =$ 7

Tableaux

Contient un nombre fixé d'éléments du même type.

- Les éléments sont rangés consécutivement en mémoire.
- Impossible de modifier le nombre d'éléments (déterminé à l'initialisation).
- Les éléments sont rangés dans des *cases* identifier par des entiers successif. Les cases d'un tableau à n éléments sont habituellement identifiés de 0 à $n-1$.
- On accède aux cases d'un tableau via les crochets [⟨*nocase*⟩].

Déclaration :
⟨variable⟩ : Tableau[⟨*debut*⟩..⟨*fin*⟩] de ⟨ type des éléments ⟩
où *debut* et *fin* sont, respectivement, l'indice de la première et la dernière case.

Exemple de trace

```
    Variables :   T : tableau[0..9] d'Entiers
                  i : entier
    Début
      Pour i de 0 à 9 faire
→       T[i] := i  mod 3
      Fin pour
    Fin
```

$T =$

0	1	2	3	4	5	6	7	8	9
0	1	2	0	1	2	0	1		

$i =$ 7

Tableaux

Contient un nombre fixé d'éléments du même type.

- Les éléments sont rangés consécutivement en mémoire.
- Impossible de modifier le nombre d'éléments (déterminé à l'initialisation).
- Les éléments sont rangés dans des *cases* identifier par des entiers successif. Les cases d'un tableau à n éléments sont habituellement identifiés de 0 à $n-1$.
- On accède aux cases d'un tableau via les crochets [⟨*nocase*⟩].

Déclaration :
⟨variable⟩ : Tableau[⟨*debut*⟩..⟨*fin*⟩] de ⟨ type des éléments ⟩
où *debut* et *fin* sont, respectivement, l'indice de la première et la dernière case.

Exemple de trace

```
    Variables :   T : tableau[0..9] d'Entiers
                  i : entier
    Début
      Pour i de 0 à 9 faire
        T[i] := i  mod 3
→     Fin pour
    Fin
```

	0	1	2	3	4	5	6	7	8	9
$T =$	0	1	2	0	1	2	0	1		

$i =$ 7

Tableaux

Contient un nombre fixé d'éléments du même type.

- Les éléments sont rangés consécutivement en mémoire.
- Impossible de modifier le nombre d'éléments (déterminé à l'initialisation).
- Les éléments sont rangés dans des *cases* identifier par des entiers successif. Les cases d'un tableau à n éléments sont habituellement identifiés de 0 à $n-1$.
- On accède aux cases d'un tableau via les crochets [⟨*nocase*⟩].

Déclaration :
⟨variable⟩ : Tableau[⟨*debut*⟩..⟨*fin*⟩] de ⟨ type des éléments ⟩
où *debut* et *fin* sont, respectivement, l'indice de la première et la dernière case.

Exemple de trace

```
    Variables :   T : tableau[0..9] d'Entiers
                  i : entier
    Début
→     Pour i de 0 à 9 faire
        T[i] := i  mod 3
      Fin pour
    Fin
```

	0	1	2	3	4	5	6	7	8	9
$T =$	0	1	2	0	1	2	0	1		

$i =$ 8

Tableaux

Contient un nombre fixé d'éléments du même type.

- Les éléments sont rangés consécutivement en mémoire.
- Impossible de modifier le nombre d'éléments (déterminé à l'initialisation).
- Les éléments sont rangés dans des *cases* identifier par des entiers successif. Les cases d'un tableau à n éléments sont habituellement identifiés de 0 à $n-1$.
- On accède aux cases d'un tableau via les crochets [⟨*nocase*⟩].

Déclaration :
⟨variable⟩ : Tableau[⟨*debut*⟩..⟨*fin*⟩] de ⟨ type des éléments ⟩
où *debut* et *fin* sont, respectivement, l'indice de la première et la dernière case.

Exemple de trace

```
    Variables :   T : tableau[0..9] d'Entiers
                  i : entier
    Début
      Pour i de 0 à 9 faire
→       T[i] := i  mod 3
      Fin pour
    Fin
```

$T =$

0	1	2	3	4	5	6	7	8	9
0	1	2	0	1	2	0	1	2	

$i = \boxed{8}$

Tableaux

Contient un nombre fixé d'éléments du même type.

- Les éléments sont rangés consécutivement en mémoire.
- Impossible de modifier le nombre d'éléments (déterminé à l'initialisation).
- Les éléments sont rangés dans des *cases* identifier par des entiers successif. Les cases d'un tableau à n éléments sont habituellement identifiés de 0 à $n-1$.
- On accède aux cases d'un tableau via les crochets [⟨*nocase*⟩].

Déclaration :
⟨variable⟩ : Tableau[⟨*debut*⟩..⟨*fin*⟩] de ⟨ type des éléments ⟩
où *debut* et *fin* sont, respectivement, l'indice de la première et la dernière case.

Exemple de trace

```
    Variables :   T : tableau[0..9] d'Entiers
                  i : entier
    Début
      Pour i de 0 à 9 faire
        T[i] := i  mod 3
→     Fin pour
    Fin
```

	0	1	2	3	4	5	6	7	8	9
$T =$	0	1	2	0	1	2	0	1	2	

$i =$ 8

Tableaux

Contient un nombre fixé d'éléments du même type.

- Les éléments sont rangés consécutivement en mémoire.
- Impossible de modifier le nombre d'éléments (déterminé à l'initialisation).
- Les éléments sont rangés dans des *cases* identifier par des entiers successif. Les cases d'un tableau à n éléments sont habituellement identifiés de 0 à $n-1$.
- On accède aux cases d'un tableau via les crochets [⟨*nocase*⟩].

Déclaration :
⟨variable⟩ : Tableau[⟨*debut*⟩..⟨*fin*⟩] de ⟨ type des éléments ⟩
où *debut* et *fin* sont, respectivement, l'indice de la première et la dernière case.

Exemple de trace

```
    Variables :   T : tableau[0..9] d'Entiers
                  i : entier
    Début
→     Pour i de 0 à 9 faire
        T[i] := i  mod 3
      Fin pour
    Fin
```

$T =$

0	1	2	3	4	5	6	7	8	9
0	1	2	0	1	2	0	1	2	

$i =$ 9

Tableaux

Contient un nombre fixé d'éléments du même type.

- Les éléments sont rangés consécutivement en mémoire.
- Impossible de modifier le nombre d'éléments (déterminé à l'initialisation).
- Les éléments sont rangés dans des *cases* identifier par des entiers successif. Les cases d'un tableau à n éléments sont habituellement identifiés de 0 à $n-1$.
- On accède aux cases d'un tableau via les crochets [⟨*nocase*⟩].

Déclaration :
⟨variable⟩ : Tableau[⟨*debut*⟩..⟨*fin*⟩] de ⟨ type des éléments ⟩
où *debut* et *fin* sont, respectivement, l'indice de la première et la dernière case.

Exemple de trace

```
    Variables :   T : tableau[0..9] d'Entiers
                  i : entier
    Début
      Pour i de 0 à 9 faire
→       T[i] := i  mod 3
      Fin pour
    Fin
```

$T =$

0	1	2	3	4	5	6	7	8	9
0	1	2	0	1	2	0	1	2	0

$i =$ 9

Tableaux

Contient un nombre fixé d'éléments du même type.

- Les éléments sont rangés consécutivement en mémoire.
- Impossible de modifier le nombre d'éléments (déterminé à l'initialisation).
- Les éléments sont rangés dans des *cases* identifier par des entiers successif. Les cases d'un tableau à n éléments sont habituellement identifiés de 0 à $n-1$.
- On accède aux cases d'un tableau via les crochets [⟨*nocase*⟩].

Déclaration :
⟨variable⟩ : Tableau[⟨*debut*⟩..⟨*fin*⟩] de ⟨ type des éléments ⟩
où *debut* et *fin* sont, respectivement, l'indice de la première et la dernière case.

Exemple de trace

```
    Variables :   T : tableau[0..9] d'Entiers
                  i : entier
    Début
      Pour i de 0 à 9 faire
        T[i] := i  mod 3
→     Fin pour
    Fin
```

$T =$

0	1	2	3	4	5	6	7	8	9
0	1	2	0	1	2	0	1	2	0

$i =$ 9

Tableaux

Contient un nombre fixé d'éléments du même type.

- Les éléments sont rangés consécutivement en mémoire.
- Impossible de modifier le nombre d'éléments (déterminé à l'initialisation).
- Les éléments sont rangés dans des *cases* identifier par des entiers successif. Les cases d'un tableau à n éléments sont habituellement identifiés de 0 à $n-1$.
- On accède aux cases d'un tableau via les crochets [⟨*nocase*⟩].

Déclaration :
⟨variable⟩ : Tableau[⟨*debut*⟩..⟨*fin*⟩] de ⟨ type des éléments ⟩
où *debut* et *fin* sont, respectivement, l'indice de la première et la dernière case.

Exemple de trace

```
    Variables :   T : tableau[0..9] d'Entiers
                  i : entier
    Début
→     Pour i de 0 à 9 faire
        T[i] := i  mod 3
      Fin pour
    Fin
```

$T =$

0	1	2	3	4	5	6	7	8	9
0	1	2	0	1	2	0	1	2	0

$i =$ 10

Tableaux

Contient un nombre fixé d'éléments du même type.

- Les éléments sont rangés consécutivement en mémoire.
- Impossible de modifier le nombre d'éléments (déterminé à l'initialisation).
- Les éléments sont rangés dans des *cases* identifier par des entiers successif. Les cases d'un tableau à n éléments sont habituellement identifiés de 0 à $n-1$.
- On accède aux cases d'un tableau via les crochets [⟨*nocase*⟩].

Déclaration :
⟨variable⟩ : Tableau[⟨*debut*⟩..⟨*fin*⟩] de ⟨ type des éléments ⟩
où *debut* et *fin* sont, respectivement, l'indice de la première et la dernière case.

Exemple de trace

Variables : T : tableau[0..9] d'Entiers
i : entier
Début
Pour i **de** 0 **à** 9 **faire**
$T[i] := i \mod 3$
Fin pour
→ **Fin**

$T =$

0	1	2	3	4	5	6	7	8	9
0	1	2	0	1	2	0	1	2	0

$i =$ 10

Les entités regroupent des éléments de types quelconques, possiblement différents. Les éléments d'une entités sont accessibles via des noms. La syntaxe pour définir une entité est :

Type ⟨nom du nouveau type⟩ : Entité
⟨Élément 1⟩ : ⟨type⟩,
⟨Élément 2⟩ : ⟨type⟩,
...
⟨Élement n⟩ : ⟨type⟩,

Étant donné une variable dont le type est une entité, on accède à ses élément de la manière suivante :

⟨variable⟩.⟨nom de l'élément⟩

Exemple :

Type : Point : Entité
x : Flottant
y : Flottant

Type : Cercle : Entité
centre : Point
rayon : Flottant

Variables : c : Cercle
p : Point
dx, dy : Flottant

Début
c.centre.x := 1.0
c.centre.y := 2.0
c.rayon := 3.0
p.x := 3.0
p.y := 4.0
dx := c.centre.x − p.x
dy := c.centre.y − p.y
Si $dx * dx + dy * dy < c$.rayon $* c$.rayon **alors**
Affiche("Le point P est dans le cercle C")
Sinon
Affiche("Le point P n'est pas dans le cercle C")
Fin si
Fin

- **Fizz-Buzz**. Écrivez le pseudo-code d'un programme qui compte de 1 à 100 en respectant les règles *Fizz-Buzz* :
  - Les multiples de 3 sont remplacés par le mot **Fizz**.
  - Les multiples de 5 sont remplacés par le mot **Buzz**.
  - Les multiples de 3 et de 5 sont remplacés par le mot **Fizz-Buzz**.

- **Ératosthène**. Écrivez un programme qui utilise le crible d'Ératosthène afin d'afficher tous les nombres premiers inférieurs à 1000000.
  - L'algorithme procède par élimination : il s'agit de supprimer d'une table des entiers de 2 à N tous les multiples d'un entier. En supprimant tous les multiples, à la fin il ne restera que les entiers qui ne sont multiples d'aucun entier, et qui sont donc les nombres premiers.
  - On commence par rayer les multiples de 2, puis à chaque fois on raye les multiples du plus petit entier restant.
  - On peut s'arrêter lorsque le carré du plus petit entier restant est supérieur au plus grand entier restant, car dans ce cas, tous les non-premiers ont déjà été rayés précédemment.
  - À la fin du processus, tous les entiers qui n'ont pas été rayés sont les nombres premiers inférieurs à N.

- **Fizz-Buzz**.

1	Fizz	11	16	⋮
2	7	Fizz	17	97
Fizz	8	13	Fizz	98
4	Fizz	14	19	Fizz
Buzz	Buzz	Fizz-Buzz	⋮	Buzz

- **Érathostène**.

2	13	31	53	⋮
3	17	37	59	999959
5	19	41	61	999961
7	23	43	67	999979
11	29	47	⋮	999983